

腾讯云小程序平台

代码接入指引

产品文档



腾讯云

【版权声明】

©2013-2024 腾讯云版权所有

本文档著作权归腾讯云单独所有，未经腾讯云事先书面许可，任何主体不得以任何形式复制、修改、抄袭、传播全部或部分本文档内容。

【商标声明】

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。

【服务声明】

本文档意在向客户介绍腾讯云全部或部分产品、服务的当时的整体概况，部分产品、服务的内容可能有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

文档目录

代码接入指引

Demo 及 SDK 获取

Android

Android SDK 说明

SDK 介绍

SDK 快速集成

SDK 集成常见问题

Android API

小程序管理 API

打开小程序

关闭小程序

删除小程序

搜索小程序

获取小程序最近访问列表

预下载小程序包

多进程交互

小程序文件管理

自定义小程序能力

自定义小程序 API

自定义小程序 API 示例

自定义小程序视图组件

定制 SDK 能力

定制小程序 UI

定制小程序 SDK 接口

定制分享能力

定制权限列表

定制权限申请

定制用户属性

定制日志及事件上报

开放接口

其他

API 说明

扩展组件 SDK

预置离线小程序

Android 机制说明

Android 常见问题

Android 错误码

Android 隐私合规

iOS

iOS SDK 说明

SDK 介绍

SDK 快速集成

SDK 集成常见问题

iOS API

小程序管理 API

搜索小程序

打开小程序

关闭小程序

删除小程序

获取小程序最近访问列表及信息

预下载小程序

小程序文件管理

预置离线小程序

自定义小程序能力

自定义小程序 API

自定义小程序视图组件

定制 SDK 能力

介绍

定制小程序 UI

定制小程序信息接口

定制分享能力

定制用户属性

日志与事件上报

开放接口

其他

API 说明

扩展组件 SDK

iOS SDK 常见问题

iOS 错误码

iOS 隐私合规

Flutter

Flutter SDK 说明

SDK 介绍

SDK 快速集成

SDK 集成常见问题

Flutter API

小程序管理 API

打开小程序

关闭小程序

删除小程序

搜索小程序

获取小程序最近访问列表

预下载小程序

自定义小程序能力

自定义小程序 API

插件定制能力

定制插件能力

定制小程序信息接口

定制分享能力

定制用户属性

事件上报

开放接口

其它

API 说明

扩展插件

Flutter 隐私合规

代码接入指引

Demo 及 SDK 获取

最近更新时间：2024-04-10 15:52:45

Android

[在线集成方式 Demo 获取](#)

[离线集成方式 Demo 及 SDK 获取](#)

iOS

说明：

按[步骤一获取配置文件](#)，替换后运行，项目 BundleId 需与运营平台设置的 BundleId 保持一致。

[通过 CocoaPods 集成 Demo 获取地址](#)

[手动集成 Demo 及 sdk 获取地址](#)

Android

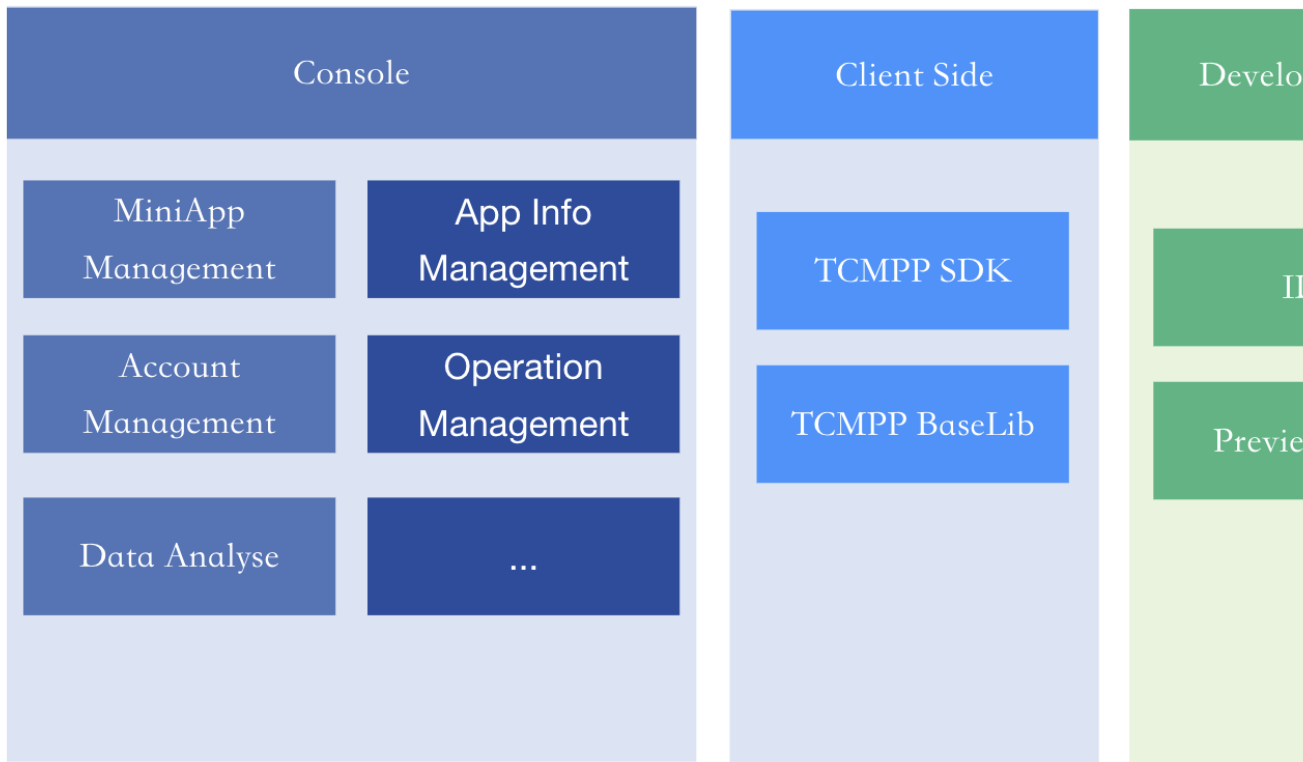
Android SDK 说明

SDK 介绍

最近更新时间：2024-06-27 11:11:57

TCMPP 介绍

TCMPP 由 管理控制台， 客户端 SDK 和 小程序开发工具（TCMPP IDE） 三个部分组成。

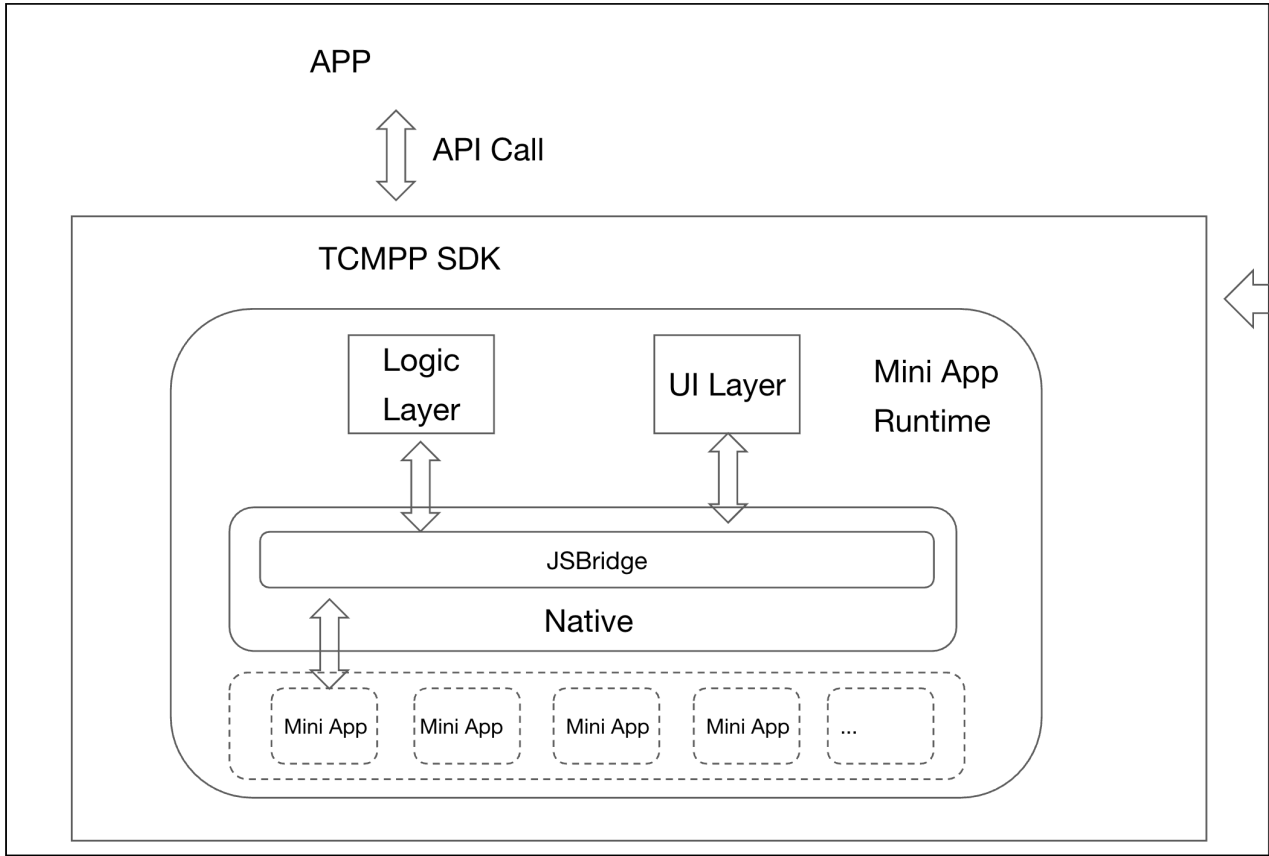


管理控制台：表示 TCMPP 控制台；包含有小程序管理、客户端应用管理、小程序运维管理等能力。

客户端：表示集成 TCMPP 小程序 SDK 的移动客户端应用程序；TCMPP 小程序 SDK 为客户端应用提供小程序的运行环境。

开发者工具：小程序开发者使用小程序 IDE 进行小程序的开发、调试和版本提交；小程序开发者使用小程序预览助手，在移动客户端进行小程序预览验证。

TCMPP 小程序 SDK 工作原理



小程序 SDK 给客户端应用提供小程序的运行时环境；小程序 SDK 通过和 TCMPP 后台通信，获取小程序信息，并在小程序 SDK 提供的运行时环境中加载和运行小程序。

小程序 SDK 的说明

为了缩小程序 SDK 的原始包体积，方便功能的可插拔，小程序 SDK 分为核心 SDK（mini_core）和小程序扩展 SDK 两个类别。

小程序核心 SDK：提供小程序加载和运行的最小环境，用户在集成小程序核心 SDK 后才能使用小程序的基础能力。

小程序扩展组件 SDK：提供更为丰富的小程序 SDK 能力，用户可以按照自己的需求进行集成。

小程序 SDK 如下表所示：

SDK 名称	功能	使用说明	包体积
小程序核心 SDK（mini_core）	提供小程序加载的最小运行时环境	-	约4.7MB
同层渲染扩展 SDK（mini_extra_static_x5_new）	提供小程序组件同层渲染的能力	需配合腾讯 X5浏览器内核及相关依赖	约54.4 KB，体积不包含腾讯X5浏览器内核

		使用，参考 扩展组件 SDK	
扫码扩展 SDK (mini_extra_qrcode)	提供扫码能力	参考 扩展组件 SDK	约35KB
地图扩展 SDK-腾讯地图版本 (mini_extra_map)	支持宿主使用腾讯地图的定位和地图服务能力	需配合腾讯地图 SDK 及相关依赖使用，参考 扩展组件 SDK	约156.4 KB，体积不包含腾讯地图 SDK
地图扩展 SDK-华为地图版本 (mini_extra_huawei_map)	支持宿主使用华为地图的定位和地图服务能力	需配合华为地图 SDK 及相关依赖使用，参考 扩展组件 SDK	约236.8 KB，体积不包含华为地图 SDK
地图扩展 SDK-Google地图版本 (mini_extra_google_map)	支持宿主使用 Google 地图的定位和地图服务能力	需配合 Google 地图 SDK 及相关依赖使用，参考 扩展组件 SDK	约153.2 KB，体积不包含 Google 地图 SDK
地图扩展 SDK-百度地图版本 (mini_extra_baidu_map)	支持宿主使用百度地图的定位和地图服务能力	需配合百度地图 SDK 及相关依赖使用，参考 扩展组件 SDK	约197 KB，体积不包含 Google 地图 SDK
地图扩展 SDK-高德地图版本 (mini_extra_amap)	支持宿主使用高德地图的定位和地图服务能力	需配合高德地图 SDK 及相关依赖使用，参考 扩展组件 SDK	约196KB，体积不包含 Google 地图 SDK
直播组件扩展 SDK(mini_extra_trtc_live)	提供小程序直播组件的能力支持	需配合腾讯 TRTC 相关依赖使用，参考 扩展组件 SDK	约91.4 KB，体积不包含腾讯 TRTC SDK
LBS 扩展 SDK(mini_extra_lbs)	提供小程序定位服务的能力支持	参考 扩展组件 SDK	约73.8 KB
蓝牙扩展 SDK(mini_extra_bluetooth)	提供小程序访问蓝牙接口的能力支持	参考 扩展组件 SDK	约115KB
NFC 扩展 SDK(mini_extra_nfc)	提供小程序访问 NFC 接口的能力支持	参考 扩展组件 SDK	约65KB
生物认证扩展 SDK(mini_extra_soter)	提供小程序访问生物认证接口的能力支持	参考 扩展组件 SDK	约71KB
剪切板扩展	提供小程序访问剪切	参考 扩展组件 SDK	约7KB

SDK(mini_extra_clipboard)	板的能力支持		
通讯录扩展 SDK(mini_extra_contact)	提供小程序访问通讯录接口的能力支持	参考 扩展组件 SDK	约49 KB
文档引擎扩展 SDK(mini_extra_doc)	提供小程序预览文档的能力支持	需配合腾讯文档 SDK 使用，参考 扩展组件 SDK	约40KB
媒体扩展 SDK(mini_extra_media_support)	提供小程序访问媒体选择接口的能力支持	参考 扩展组件 SDK	约568.1 KB

SDK 快速集成

最近更新时间：2024-06-27 11:11:46

快速集成将指导客户端开发人员集成一个最小化功能的小程序 SDK，实现小程序的打开功能。

开始集成

1. 配置在线仓库地址

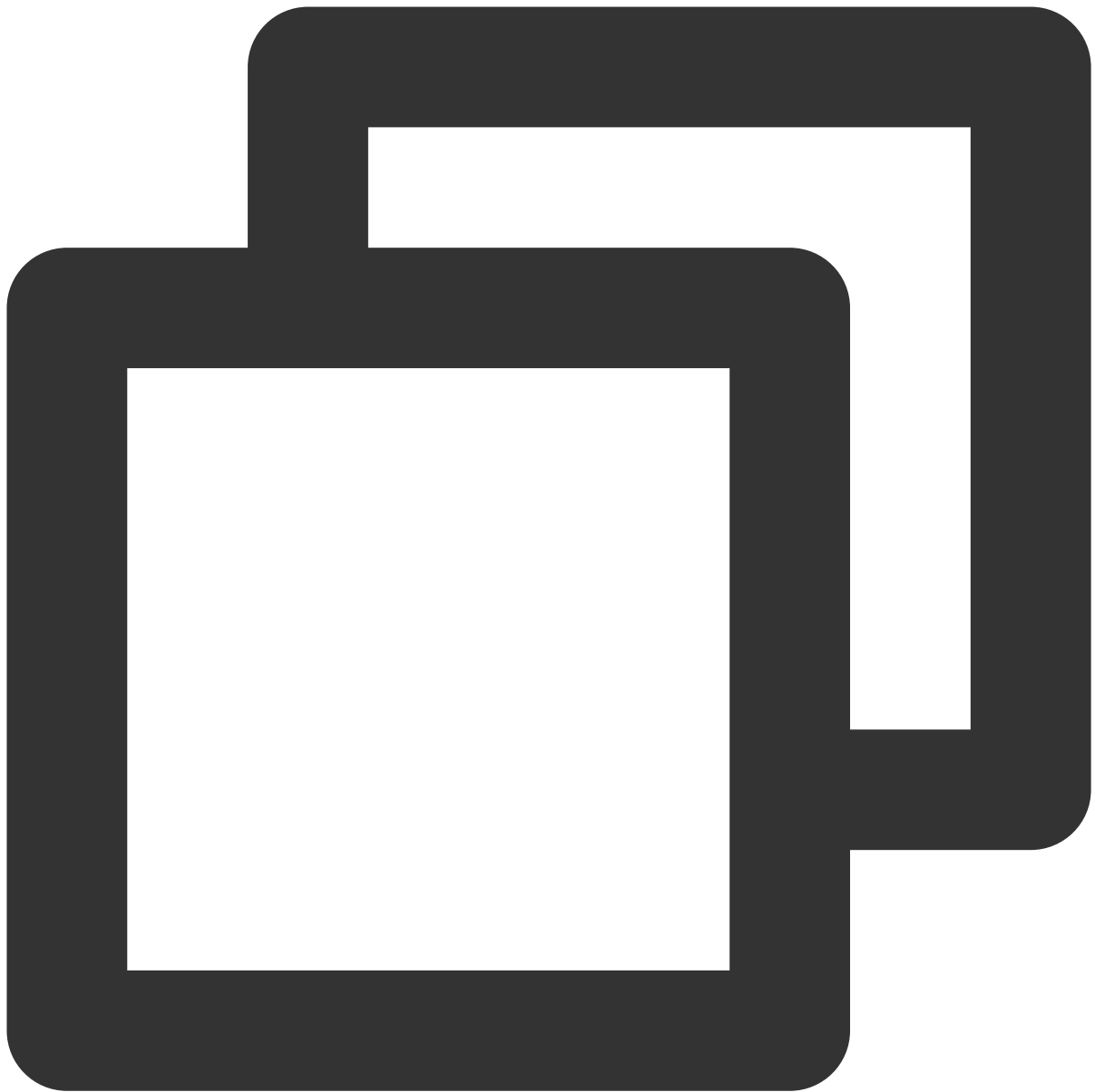
仓库地址的配置方式和当前项目中所使用的gradle插件版本相关；

gradle 插件的版本为 Gradle 7.0以前版本的，参考方式一配置仓库信息。

gradle 插件的版本为 Gradle 7.1及以后版本的，参考方式二配置仓库信息。

方式一：Gradle 7.0以前版本配置。

打开用户项目管理工具，在项目级 `build.gradle` 中添加如下配置。



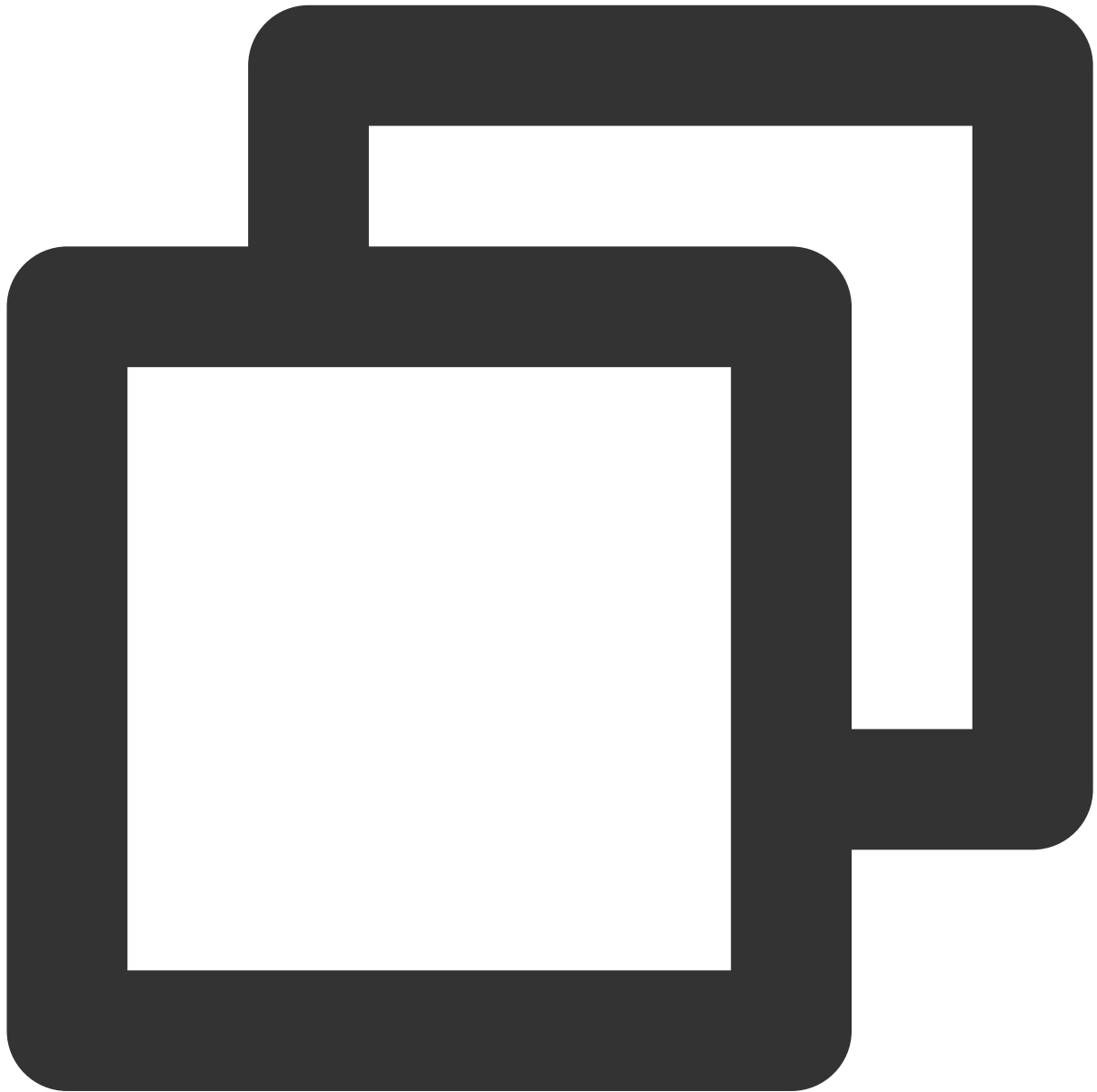
```
buildscript {  
    dependencies {  
        classpath 'org.jetbrains.kotlin:kotlin-gradle-plugin:1.4.32'  
    }  
}  
  
allprojects {  
    repositories {  
        maven {  
            url 'https://tcmpp-work-maven.pkg.coding.net/repository/tcmpp/android'  
        }  
    }  
}
```



```
}  
}
```

方式二：Gradle 7.1及之后版本配置。

在项目级 `settings.gradle` 中添加如下配置。



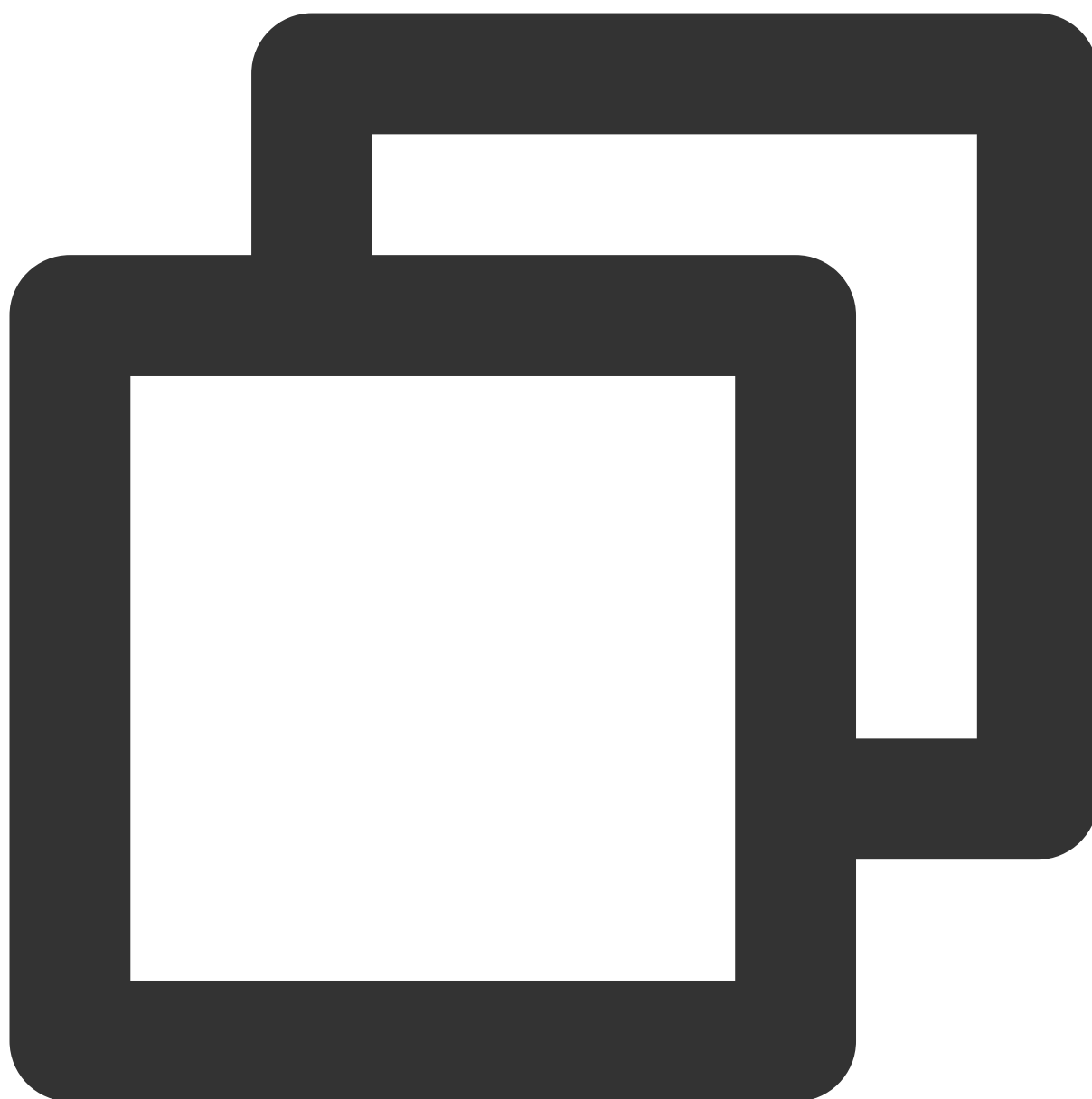
```
pluginManagement {  
    repositories {  
        maven {  
            url 'https://tcmpp-work-maven.pkg.coding.net/repository/tcmpp/android'  
        }  
    }  
}
```

```
    }  
}  
dependencyResolutionManagement {  
    repositories {  
        maven {  
            url 'https://tcmpp-work-maven.pkg.coding.net/repository/tcmpp/android'  
        }  
    }  
}
```

2. kotlin 插件配置

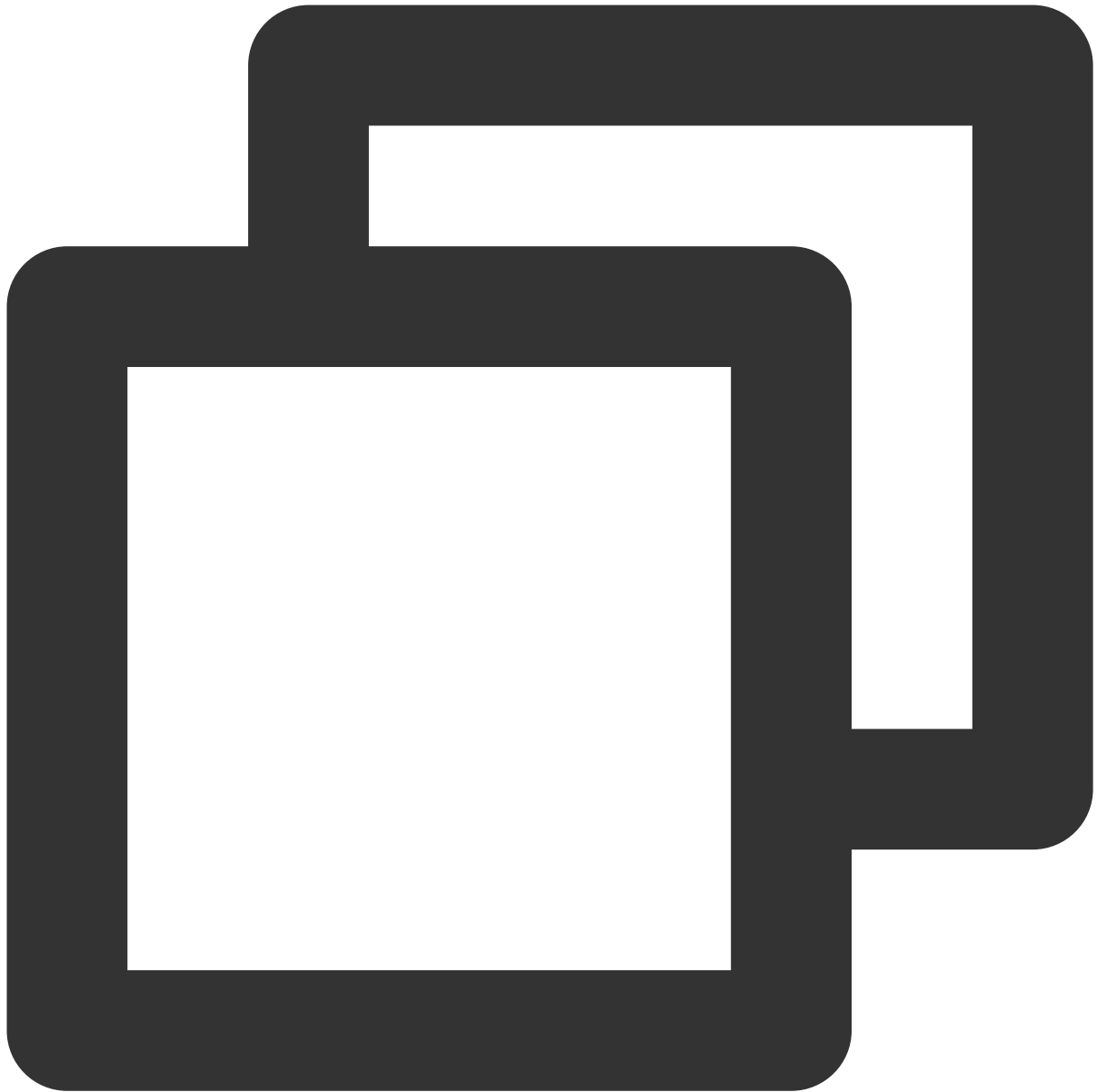
小程序 SDK 的配置同样需要根据 **gradle** 插件的版本进行配置，如下：

方式一：Gradle 7.0以前版本。



```
apply plugin: 'kotlin-android'  
apply plugin: 'kotlin-kapt'
```

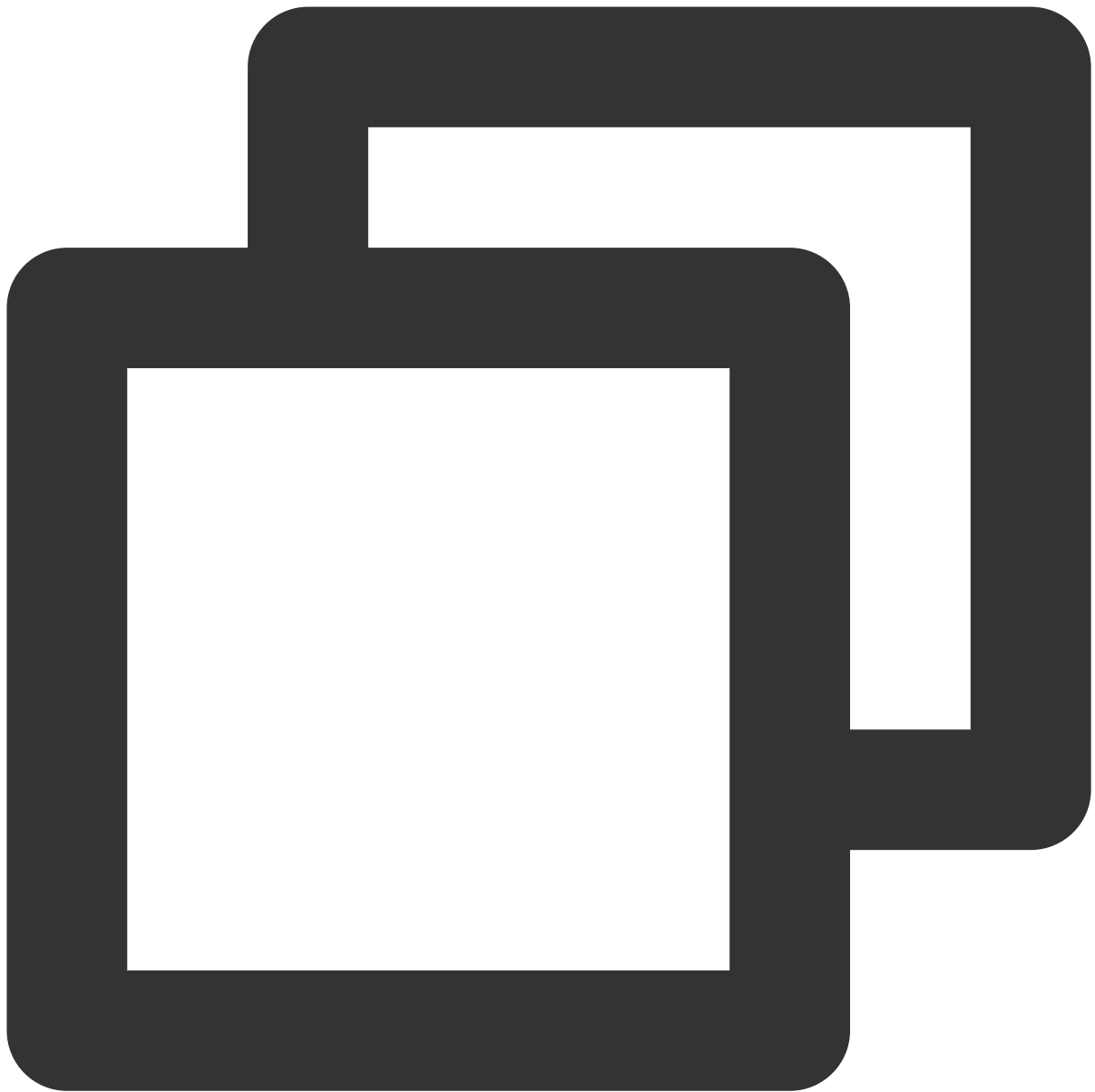
方式二：Gradle 7.1及之后版本。



```
plugins {  
    id "org.jetbrains.kotlin.android"  
    id "org.jetbrains.kotlin.kapt"  
}
```

3. 配置 `packagingOptions`

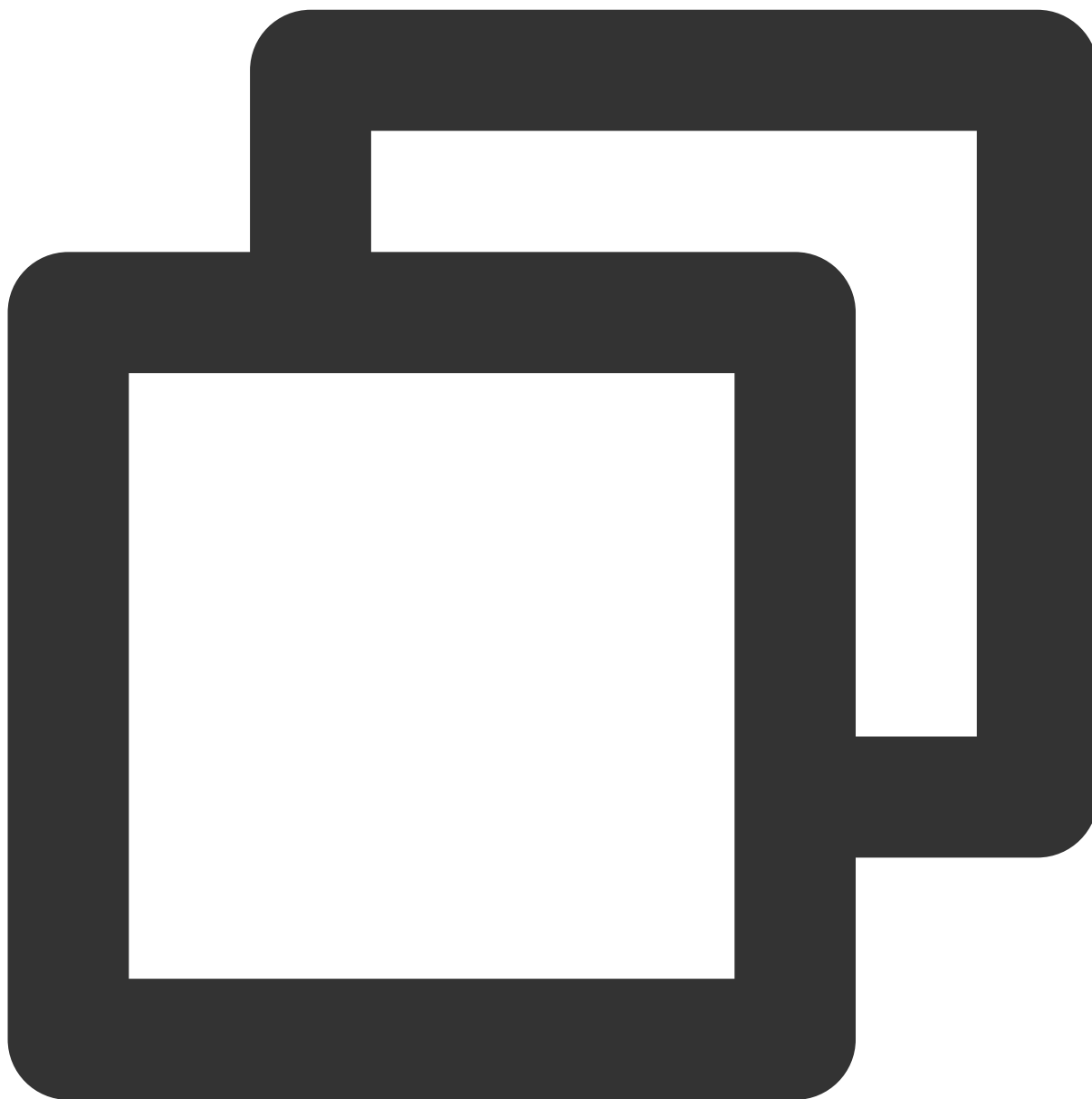
打开应用级别的 `build.gradle` 文件，增加如下配置。



```
android {  
  defaultConfig {  
    packagingOptions {  
      pickFirst 'lib/arm64-v8a/libc++_shared.so'  
      pickFirst 'lib/armeabi/libc++_shared.so'  
      pickFirst 'lib/armeabi-v7a/libc++_shared.so'  
      pickFirst 'lib/arm64-v8a/libmarsxlog.so'  
      pickFirst 'lib/armeabi/libmarsxlog.so'  
      pickFirst 'lib/armeabi-v7a/libmarsxlog.so'  
      pickFirst 'lib/arm64-v8a/libv8jni.so'  
    }  
  }  
}
```

```
}  
}
```

4.配置小程序 SDK 的依赖



```
dependencies {  
    implementation 'com.google.android.material:material:1.3.0-alpha03'  
    implementation 'androidx.core:core-ktx:1.6.0'  
  
    //gosn
```

```
implementation 'com.google.code.gson:gson:2.8.6'
// ok-http
implementation 'com.squareup.okhttp3:okhttp:3.12.13'

// mini app start
kapt 'com.tencent.tcmpp.android:mini_annotation_processor:${version}'//版本信息请
implementation 'com.tencent.tcmpp.android:mini_core:${version}'//版本信息请参考SDI
// mini app end
}
```

说明：

配置中的 {version} 需要替换为对应依赖的版本；版本信息参考 [Android 更新日志](#)。

如果开发者原来的工程中使用了 `annotationProcessor` 注解处理器，需要将所有 `annotationProcessor` 改为 `kapt` 注解器。

如需使用同层渲染的能力，需要集成 x5 内核，可参考 [扩展组件 SDK](#)。

小程序 SDK 的依赖引入完毕后，可以开始进行小程序 SDK 的接入工作，如下面章节所示。

接入前准备工作

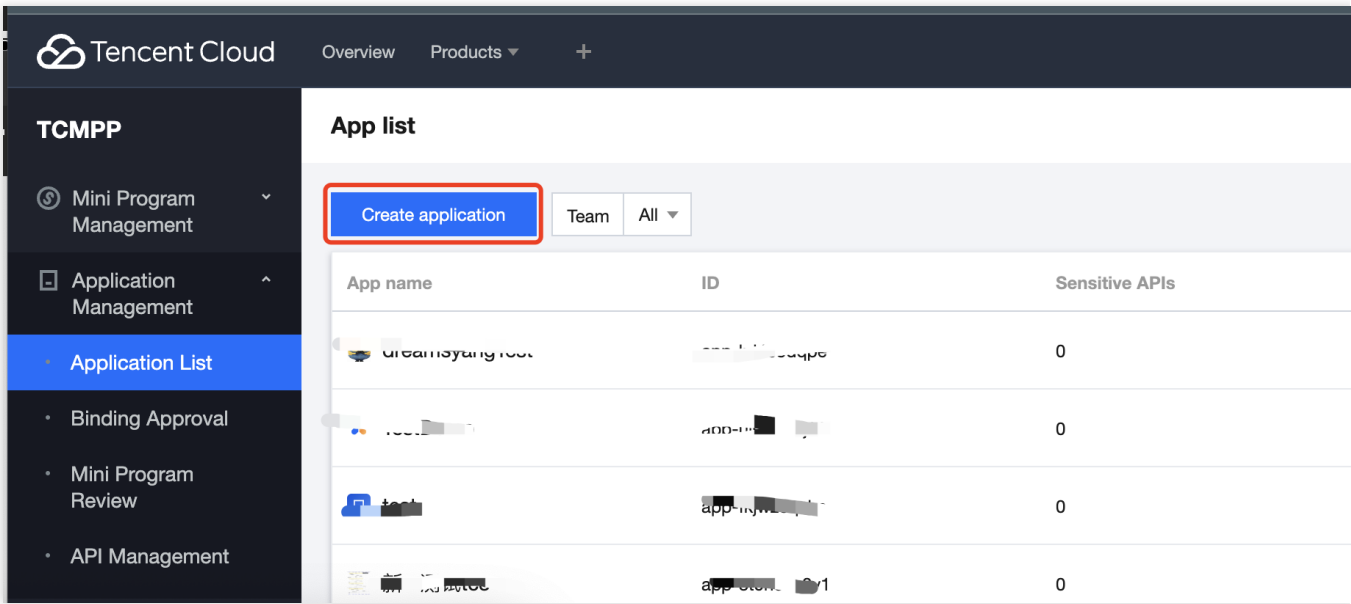
1. 获取配置文件

小程序 SDK 的初始化依赖于从 `小程序控制台` 获取的 `小程序 SDK 配置文件`；在开始集成小程序 SDK 前，需要从小程序控制台获取小程序 SDK 配置文件。

获取配置文件过程如下：

登陆小程序控制台

创建应用



填写应用信息

Step 1: Fill in the app information

1 Create application >

2 Integrate mini program container

Team *

test

App name *

MySuperApp

Supports 3-64 characters consisting of Chinese characters, uppercase and lowercase letters, numbers, spaces, and some

App overview

description

App icon



No image is selected.

Select image

Please upload a square image in .jpg and .png format with a resolution of 128*128. The image is less than 2 MB.
If the icon is not uploaded, the system default icon will be used.

Integration platform *



iOS platform



Android Platform

Package Name *

com.your.package.name

App download address

Please enter the app download a

Next Step, Integrate the Mini Program Container

下载配置文件

注意：

注意：默认下载的配置文件的名称为：tcmpp-android-configurations.json

[←](#) Create application

Step 2: Integrate the mini program container and programs.

✓ Create application > 2 Integrate mini program container

1 Specify the app package name and obtain the configuration file.

Package Name com.your.package.name

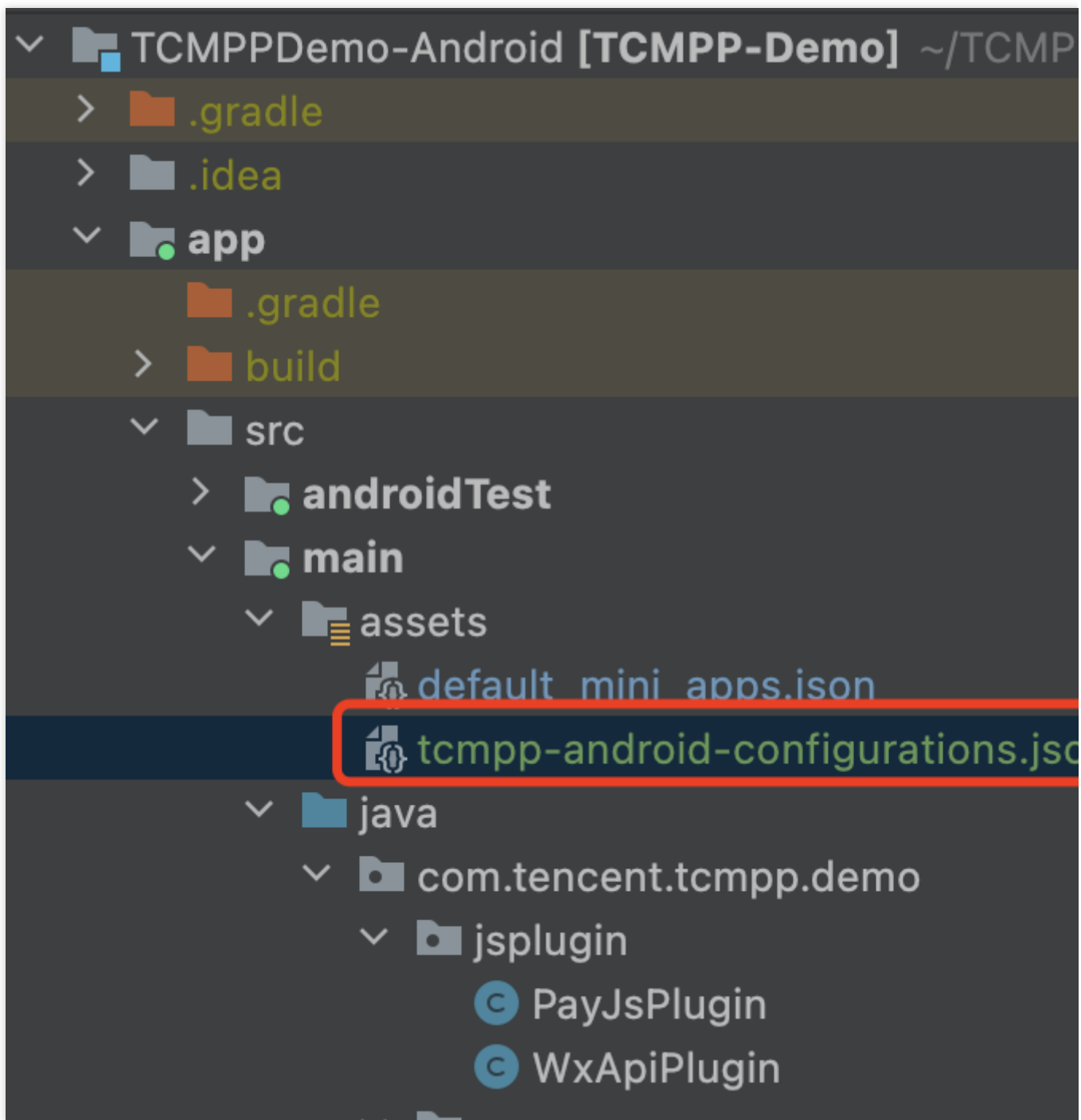
Configuration file ⓘ

[Download configuration file](#)

2 Import the configuration file into the assets root directory of the application.

2.添加配置文件到项目

获取配置文件后，需要将 配置文件 拷贝到应用程序源码工程的 assets 路径中：

**注意：**

应用包名必须与配置文件中 `packageName` 保持一致，否则小程序 SDK 在运行时无法通过包名校验，导致 SDK 初始化异常。

如果不一致，不能直接修改配置文件包名，应通过下面两种方式来修正：

修改工程中应用包名，与配置文件中包名一致。

控制台修改应用包名，与工程中应用包名保持一致，并重新下载配置文件。

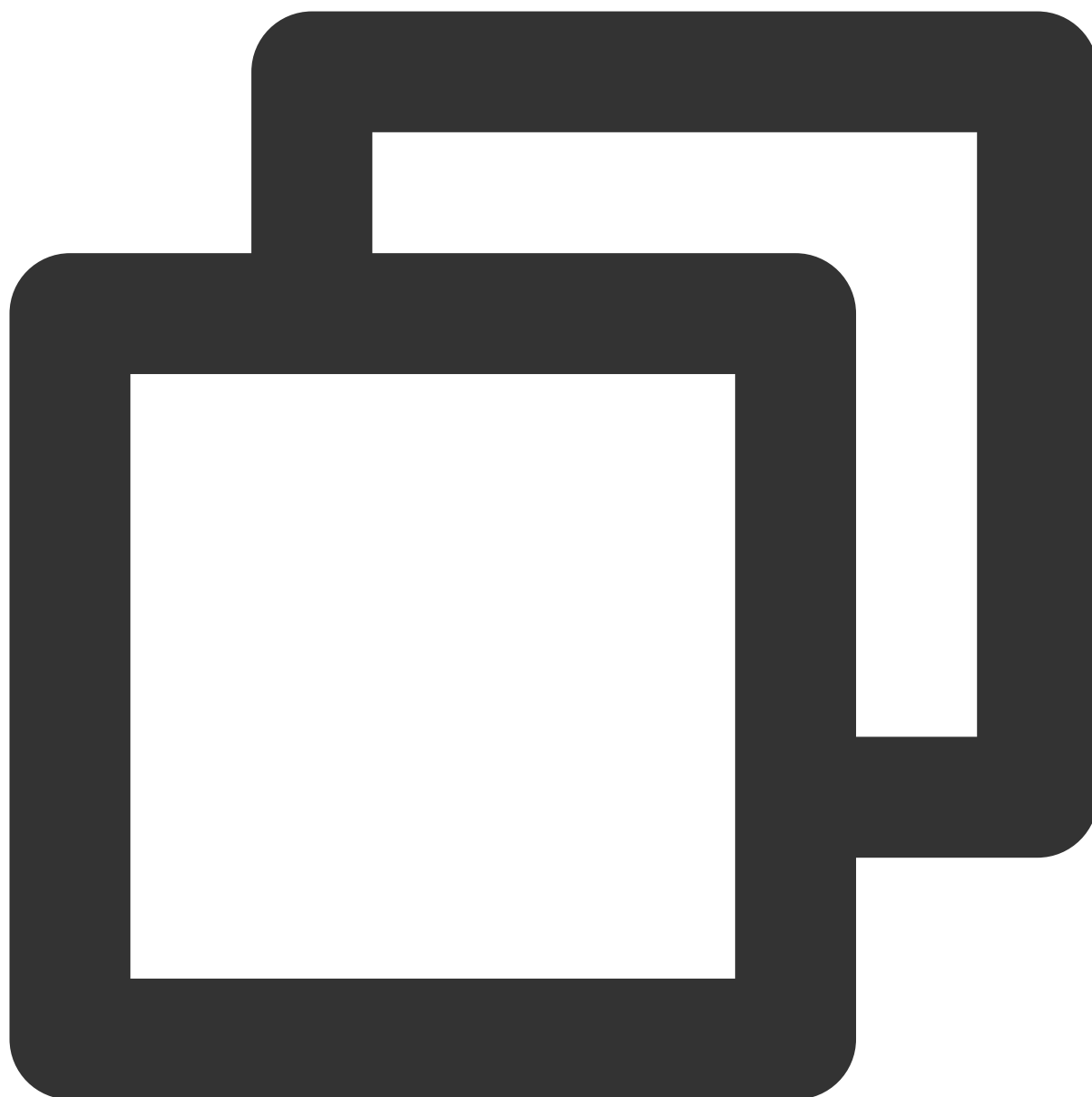
配置文件中的 `packageName` 字段，需要和当前项目的包名一致。

```
{
  "customId": "T1682645488JDYWMH",
  "productId": "7422",
  "packageName": "com.tencent.tcmpp.showcase",
  "bundleId": "",
  "material": "01rg5oB0lgkX9bGM7/C8k7RBovDV4ByAGr",
  "shark": {
    "httpUrl": "https://api.tcmpp-staging.tmfcloud",
    "tcpHost": "api.tcmpp-staging.tmfcloud.com",
    "tcpPort": 30014,
    "appKey": "app-1lwbqz2nop",
    "symEnc": 2,
    "asymEnc": 1
  }
}
```

3.源码中增加小程序 SDK 的初始化配置

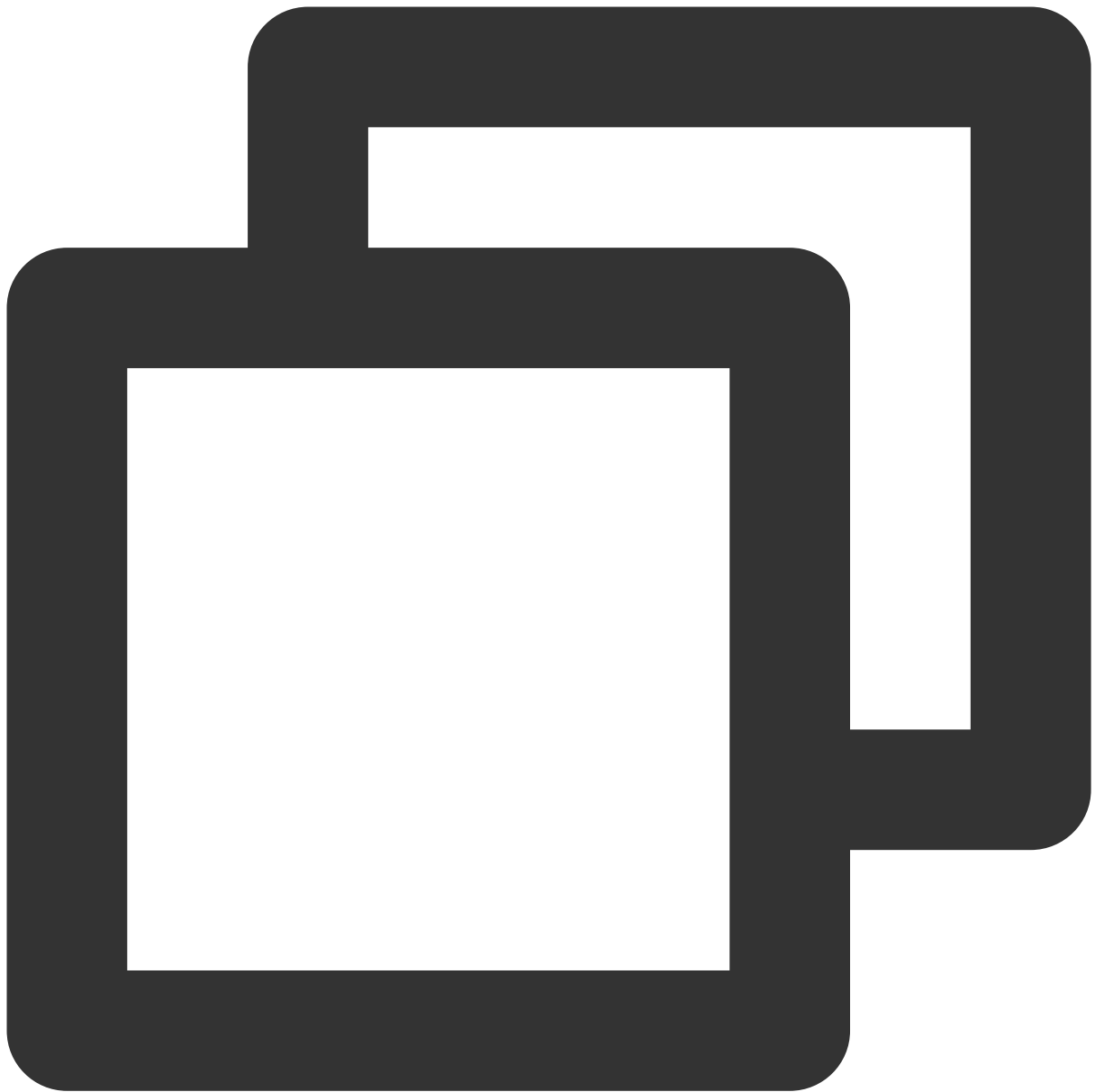
继承实现 MiniConfigProxy 类；

MiniConfigProxy 的实现类，添加注解如下：



```
@ProxyService(proxy = MiniConfigProxy.class)
```

参考代码如下：



```
@ProxyService(proxy = MiniConfigProxy.class)
public class MiniConfigProxyImpl extends MiniConfigProxy {
    /**
     * 应用Application
     * @return
     */
    @Override
    public Application getApp() {
        return "app Application";
    }
}
```

```
/**
 * 创建初始化配置信息
 * @return
 */
@Override
public MiniInitConfig buildConfig() {
    MiniInitConfig.Builder builder = new MiniInitConfig.Builder();
    MiniInitConfig config = builder
        .configAssetName("tcmpp-android-configurations.json") //assets中配置
        .imei("IMEI") //配置设备id, 用于在管理平台上根据设备标识进行小程序的灰度发布
        .autoRequestPermission(true) //配置小程序使用到需要权限的API时是否自动向用户
        .debug(true) //日志开关, 默认关闭的
        .build();
}
```

注意：

需要注意的是，MiniConfig 的 configAssetName 用于指定配置文件在源码工程中的路径和名称。

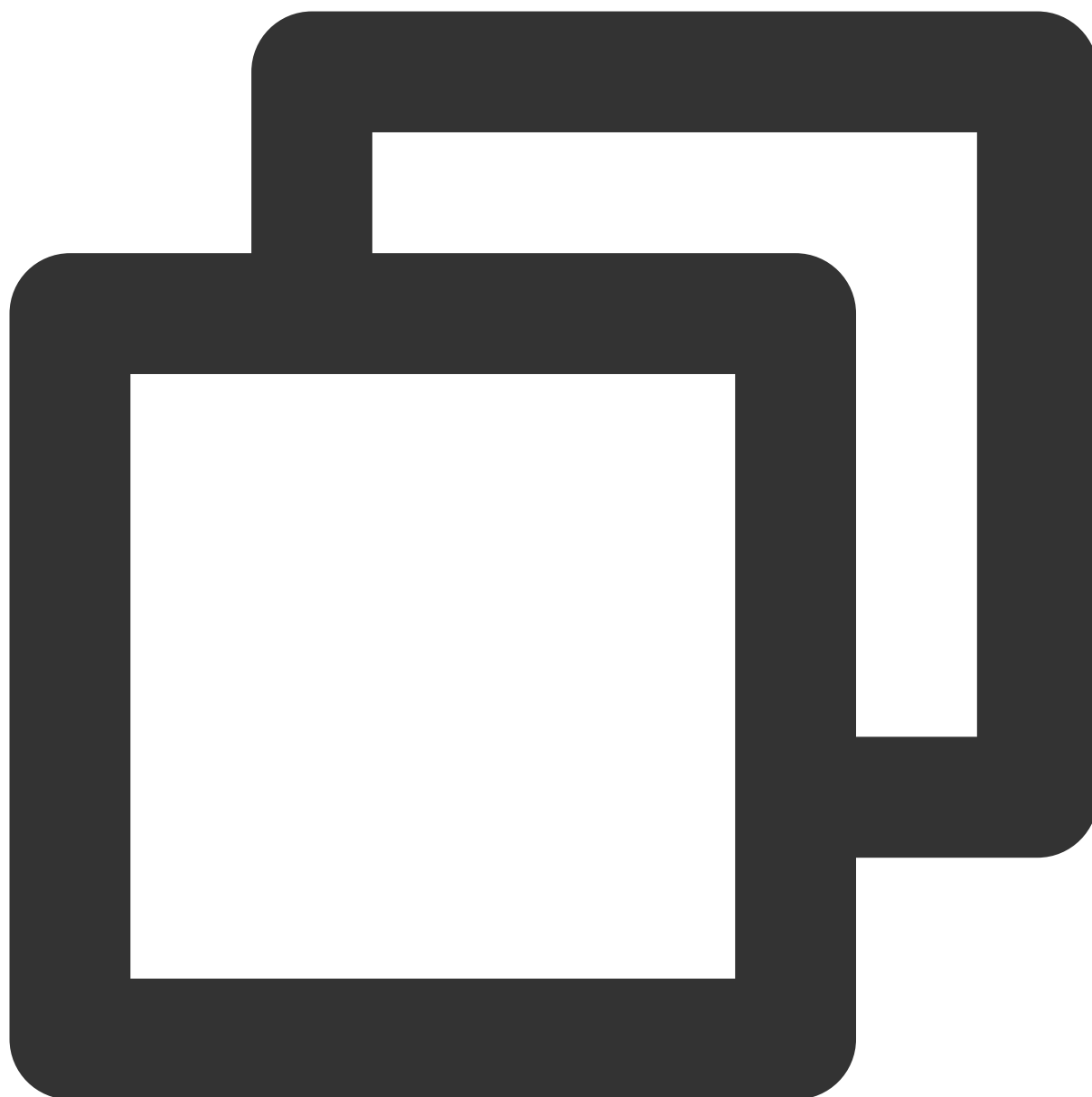
至此，小程序SDK的接入就完成了。下一步，可以通过小程序 SDK 提供的 API 实现小程序的打开和预览。

打开小程序

打开小程序可以参考如下的实例代码：

说明：

appId 为小程序 ID，需要在小程序开发人员获取。

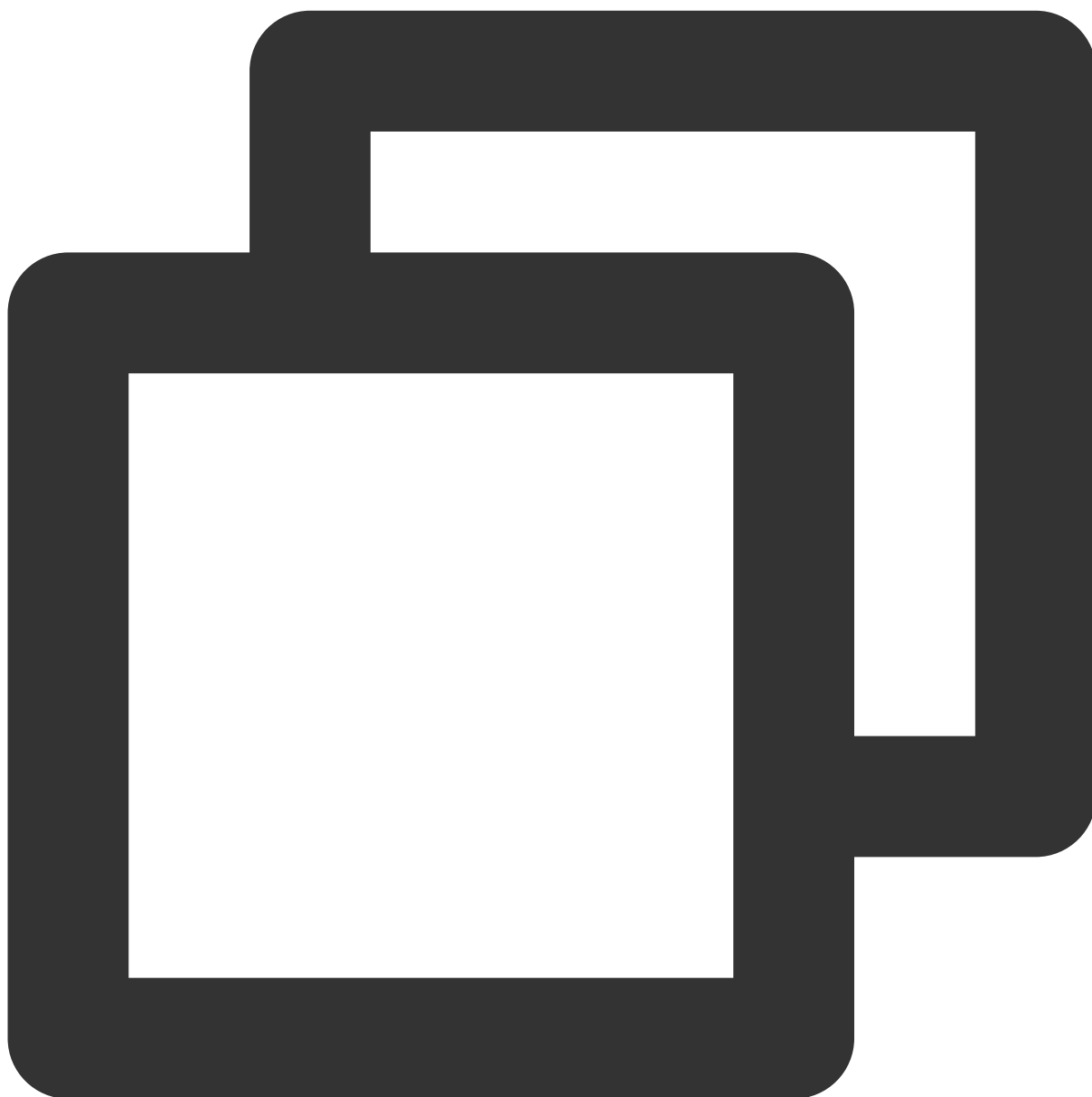


```
TmfMiniSDK.startMiniApp(activity, appId, miniStartOptions);
```


SDK 集成常见问题

最近更新时间：2024-06-27 11:10:14

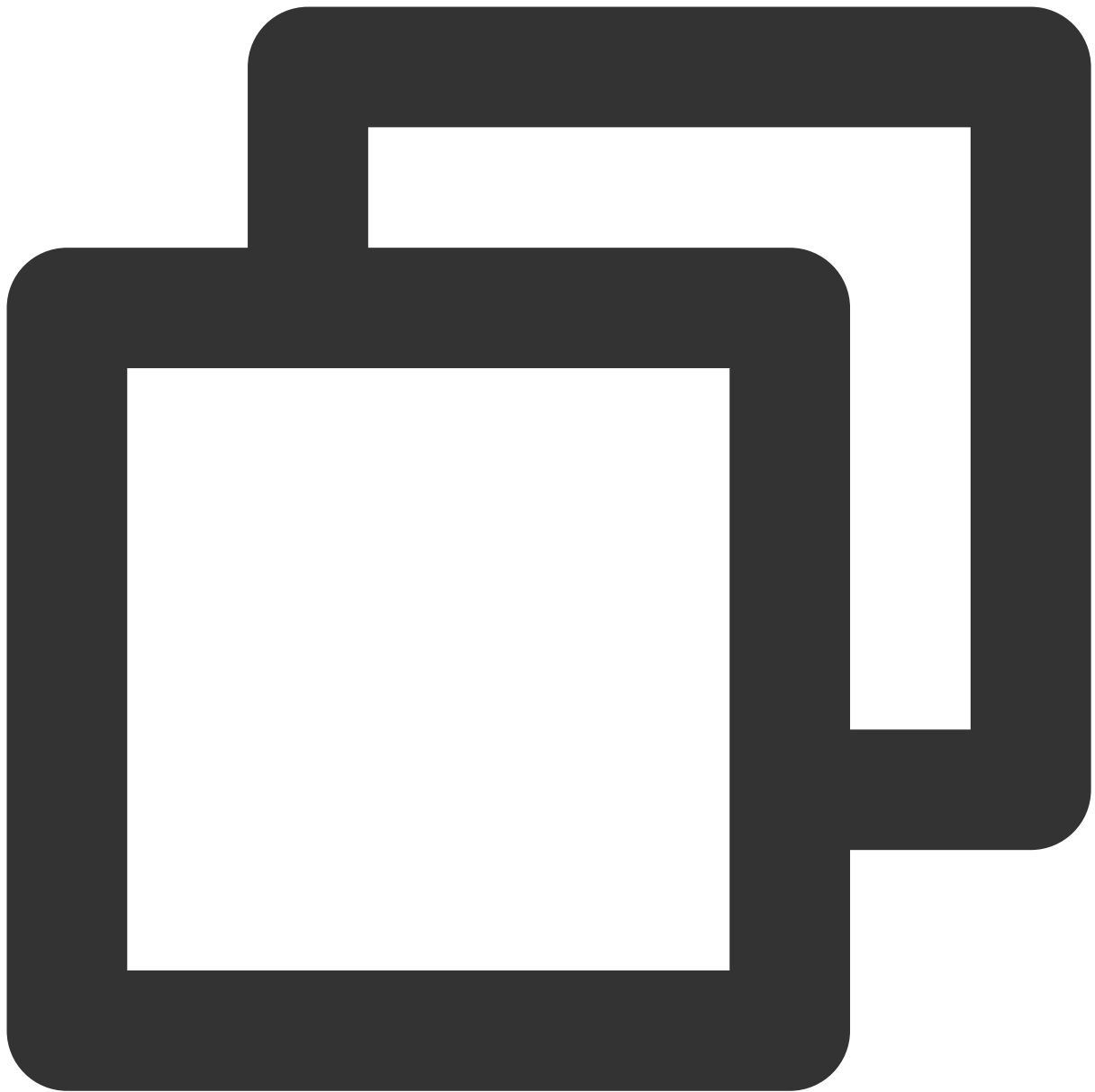
接入小程序 SDK 后，执行项目编译构建时，出现如下 AAPT 问题



```
AAPT: error: attribute android:requestLegacyExternalStorage not found.
```

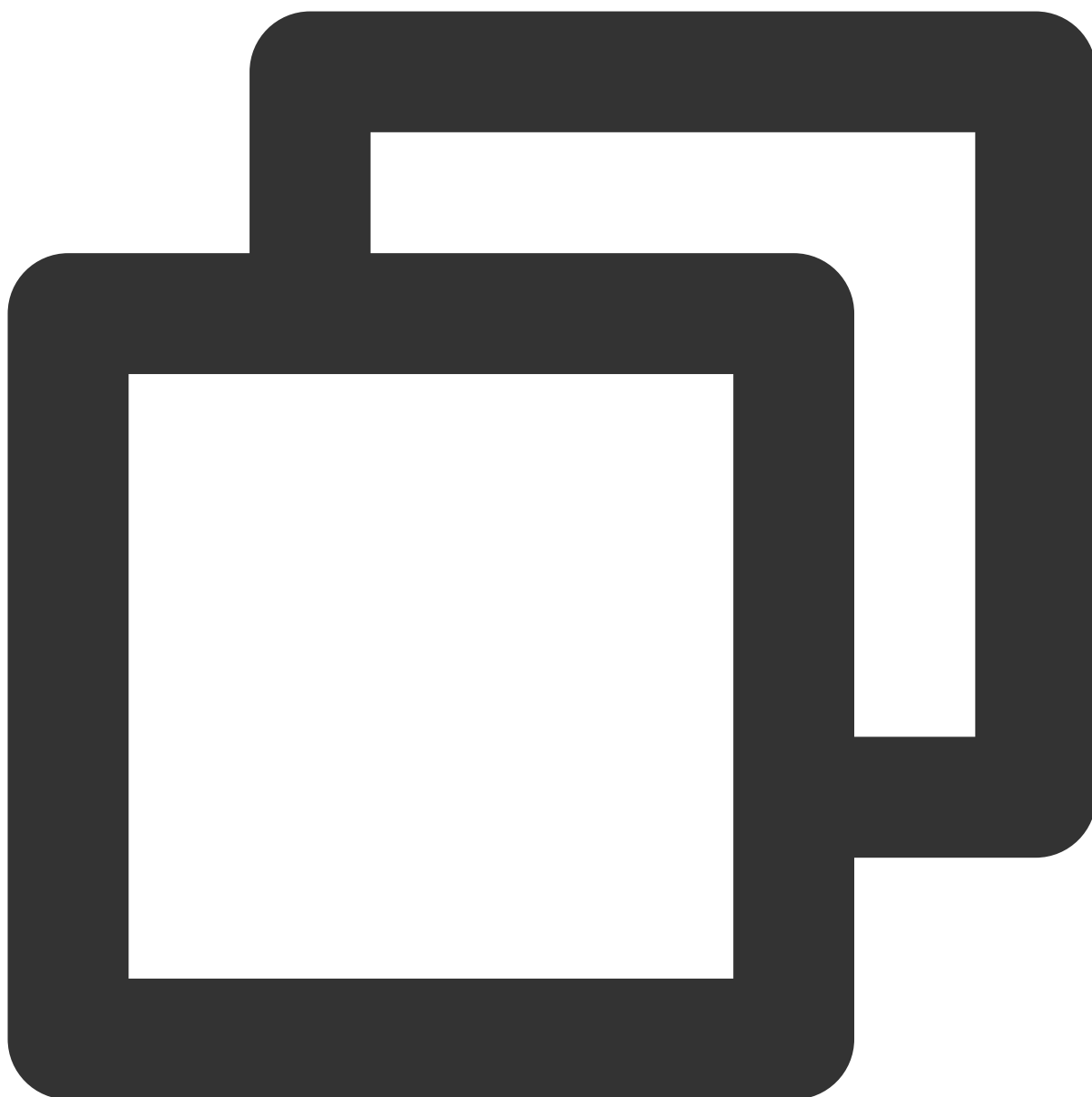
解决方法：

在 `AndroidMainifest.xml` 的 `<application>` 标签下添加如下配置：



```
<application
  android:theme="@style/AppTheme"
  tools:replace="android:icon"
  tools:remove="android:requestLegacyExternalStorage">
/application>
```

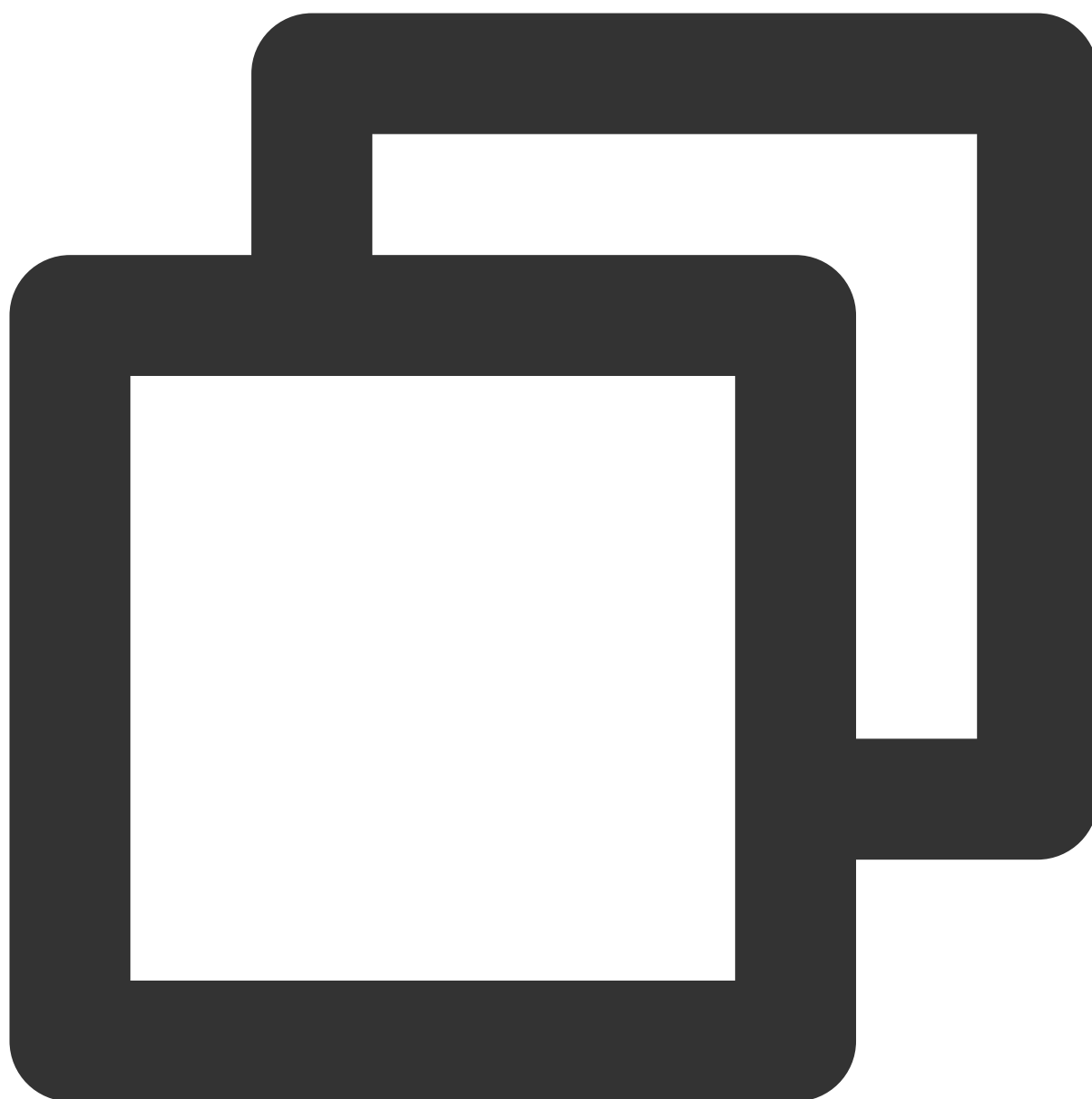
接入小程序 SDK 后，执行项目编译构建时，出现如下“Duplicate class android.support.v4”问题



```
Duplicate class android.support.v4.app.INotificationSideChannel found in modules co
```

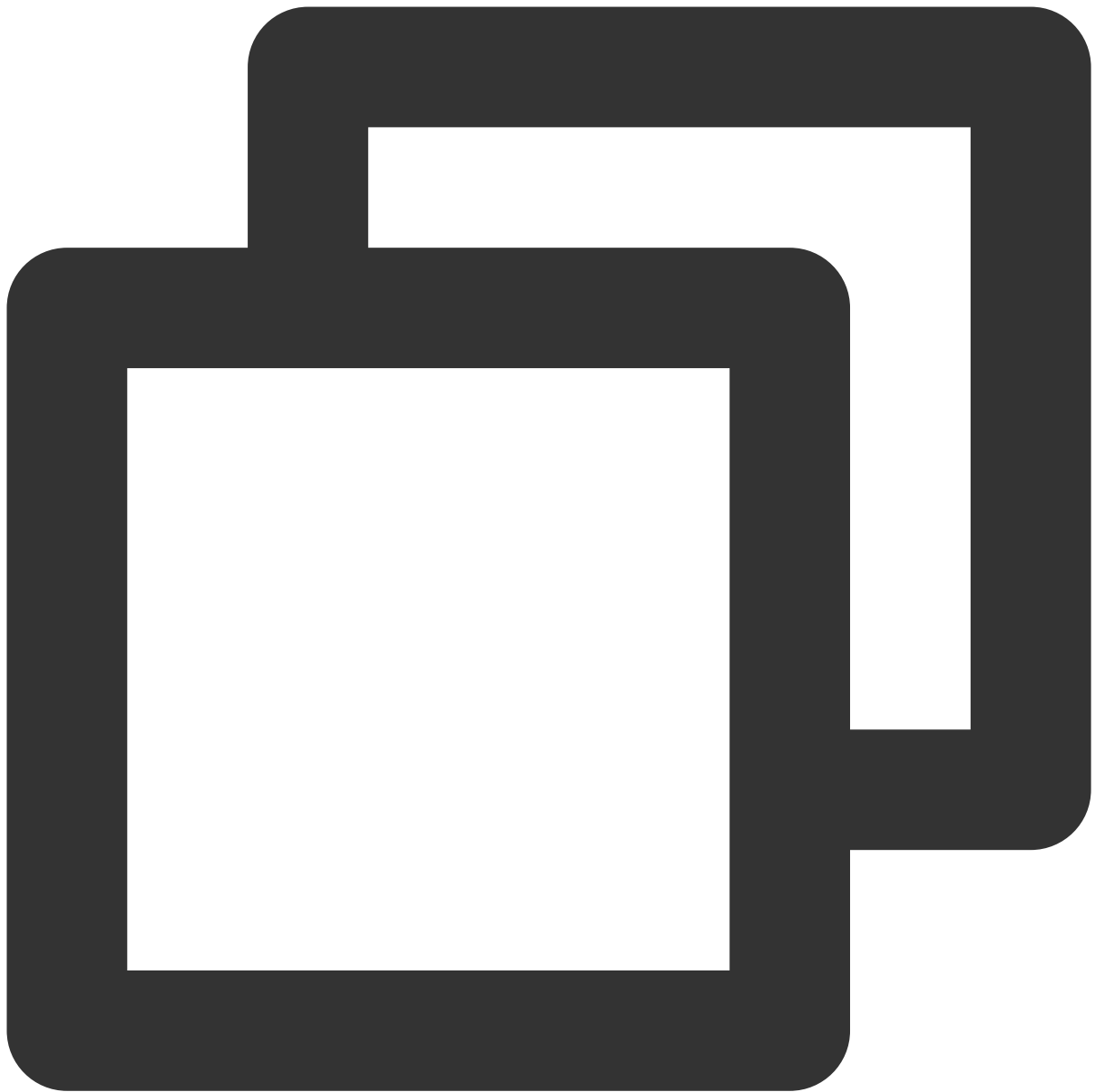
解决方法：

在 `gradle.properties` 中添加如下代码：



```
android.useAndroidX=true  
android.enableJetifier=true
```

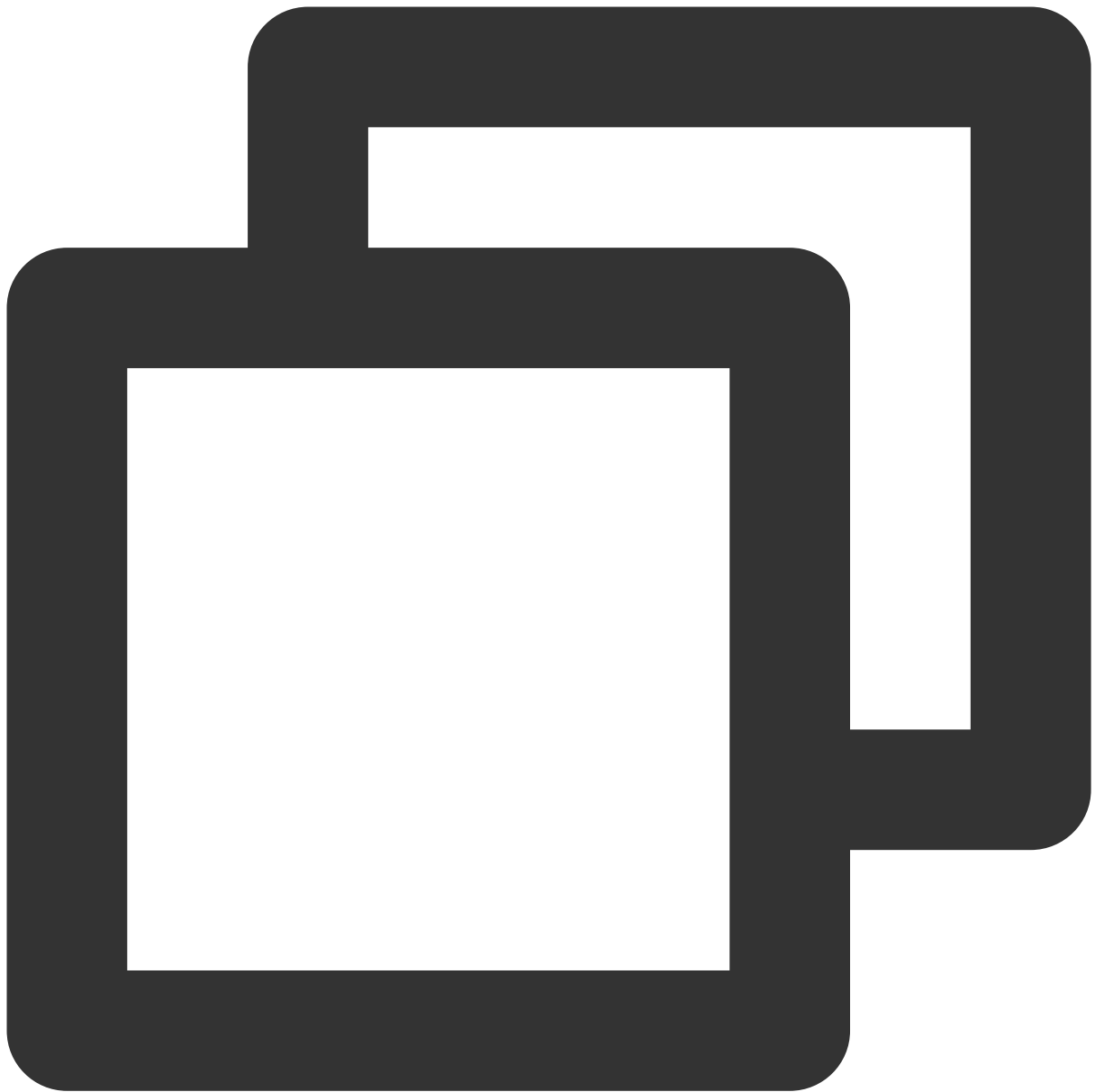
接入小程序 SDK 后，执行项目编译构建时，出现如下“`compileDebugJavaWithJavac`”版本匹配问题：



```
Execution failed for task ':app:kaptGenerateStubsDebugKotlin'.
> 'compileDebugJavaWithJavac' task (current target is 1.8) and 'kaptGenerateStubsDe
Consider using JVM toolchain: https://kotlin.in/gradle/jvm/toolchain
```

解决方法：

将build.gradle中的compileOptions中的JDK版本修改为合适版本。



```
android {  
  
    compileOptions {  
        sourceCompatibility JavaVersion.VERSION_17 //需要匹配kapt版本  
        targetCompatibility JavaVersion.VERSION_17 //需要匹配kapt版本  
    }  
}
```

接入小程序 SDK 后，执行项目编译构建时，出现如下“`java.lang.NoClassDefFound`，`ProxyService`”问题，如下图

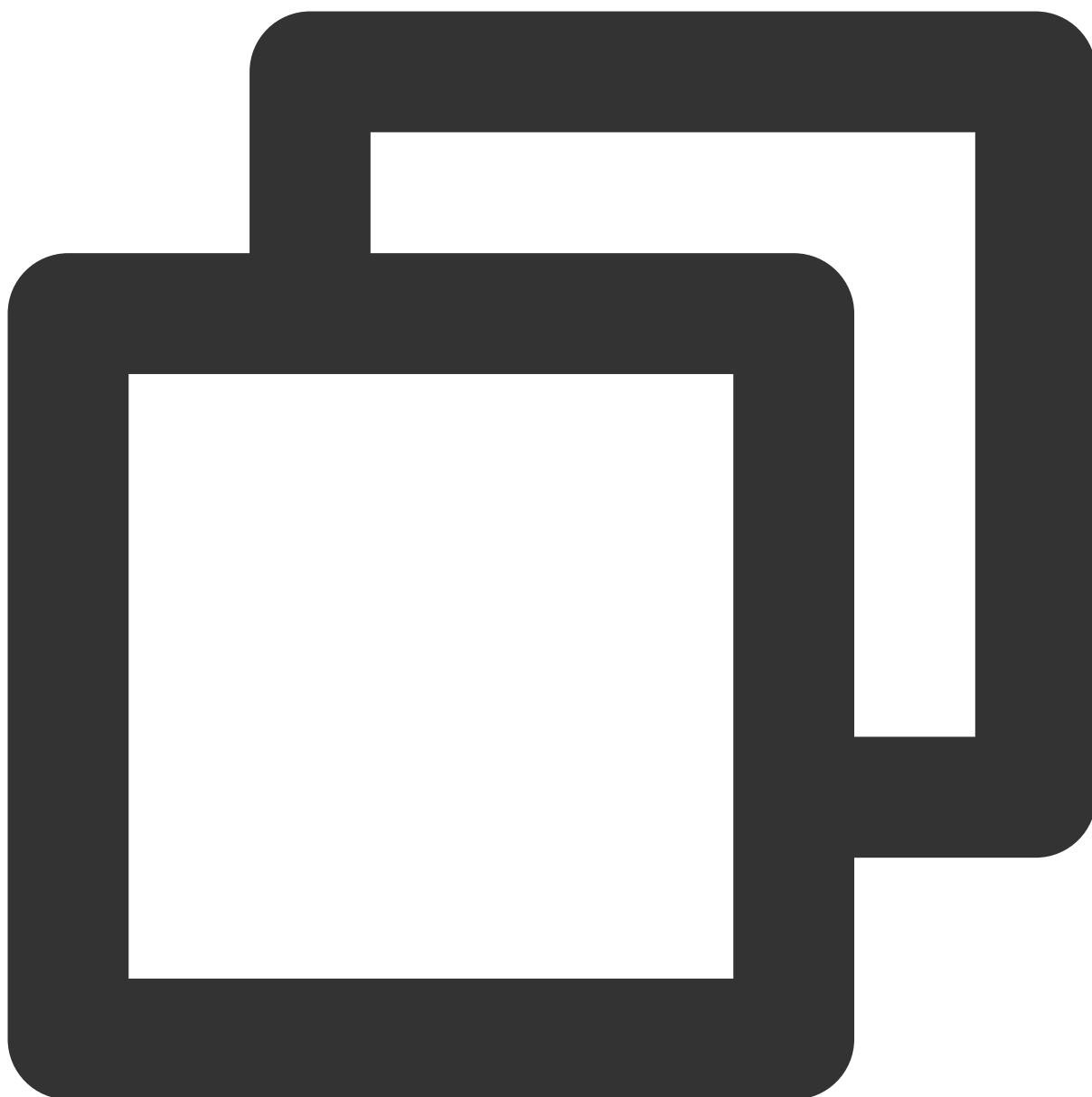
```
> A failure occurred while executing org.jetbrains.kotlin.gradle.internal.KaptWithoutKotlincTask$KaptExec
> java.lang.reflect.InvocationTargetException (no error message)

* Try:
> Run with --info or --debug option to get more log output.
> Run with --scan to get full insights.

* Exception is:
org.gradle.api.tasks.TaskExecutionException: Execution failed for task ':libs:libopensdk:kaptEnvTestKotli
Caused by: org.gradle.workers.internal.DefaultWorkerExecutor$WorkExecutionException: A failure occurred w
org.jetbrains.kotlin.gradle.internal.KaptWithoutKotlincTask$KaptExecutionWorkAction <126 internal lines>
Caused by: java.lang.reflect.InvocationTargetException <3 internal lines>
    at org.jetbrains.kotlin.gradle.internal.KaptExecution.run(KaptWithoutKotlincTask.kt:285)
    at org.jetbrains.kotlin.gradle.internal.KaptWithoutKotlincTask$KaptExecutionWorkAction.execute(KaptWi
    <27 internal lines>
    ... 2 more
Caused by: java.lang.reflect.InvocationTargetException <3 internal lines>
    at org.jetbrains.kotlin.kapt3.base.AnnotationProcessingKt.doAnnotationProcessing(annotationProcessing
    at org.jetbrains.kotlin.kapt3.base.AnnotationProcessingKt.doAnnotationProcessing$default(annotationPr
    at org.jetbrains.kotlin.kapt3.base.Kapt.kapt(Kapt.kt:47)
    ... 34 more
Caused by: com.sun.tools.javac.processing.AnnotationProcessingError Create breakpoint : java.lang.NoClassDefF
com.tencent/tmfmini/sdk/annotation/ProxyService
    at jdk.compiler/com.sun.tools.javac.processing.JavacProcessingEnvironment.callProcessor(Unknown Source
    at jdk.compiler/com.sun.tools.javac.processing.JavacProcessingEnvironment.discoverAndRunProcs(Unknown
    at jdk.compiler/com.sun.tools.javac.processing.JavacProcessingEnvironment$Round.run(Unknown Source)
    at jdk.compiler/com.sun.tools.javac.processing.JavacProcessingEnvironment.doProcessing(Unknown Source
    at jdk.compiler/com.sun.tools.javac.main.JavaCompiler.processAnnotations(Unknown Source)
    ... 40 more
Caused by: java.lang.NoClassDefFoundError: com.tencent/tmfmini/sdk/annotation/ProxyService
    at com.tencent.tmfmini.sdk.annotation.processor.ProxyServiceProcessor.process(ProxyServiceProcessor.j
    at org.jetbrains.kotlin.kapt3.base.incremental.IncrementalProcessor.process(incrementalProcessors.kt:
    at org.jetbrains.kotlin.kapt3.base.ProcessorWrapper.process(annotationProcessing.kt:188)
    ... 45 more
Caused by: java.lang.ClassNotFoundException: com.tencent.tmfmini.sdk.annotation.ProxyService
    ... 48 more
```

解决方法：

检查工程中是否有如下配置，如果有则去掉。



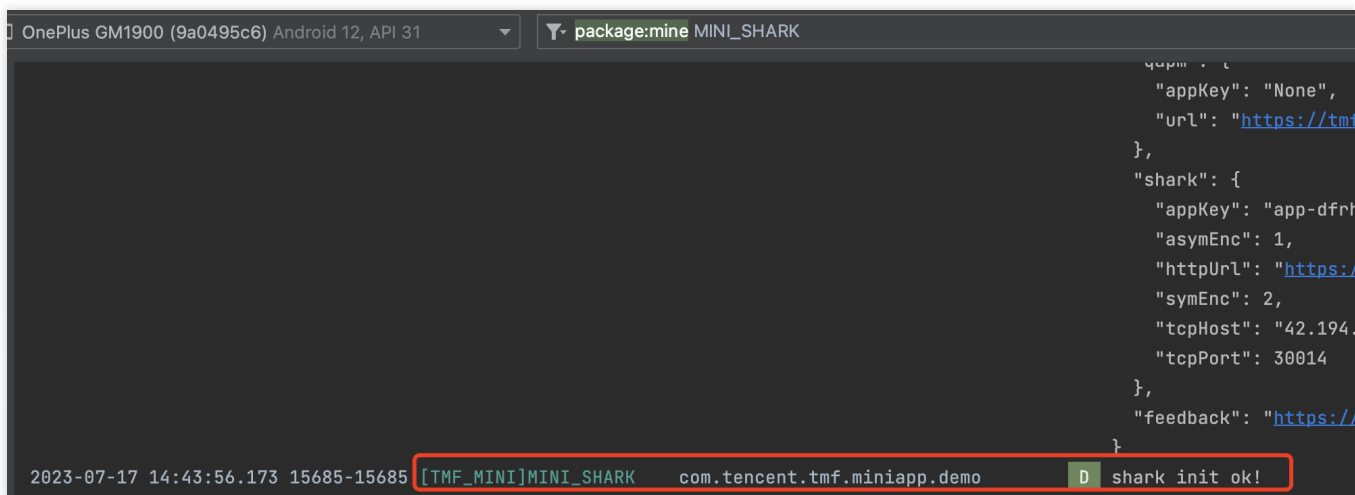
```
kapt.include.compile.classpath=false
```

小程序 SDK 支持的最小 Android 系统版本是多少？

最小 SDK 版本（minSdkVersion）：21，对应 Android 版本为Android 5.0。

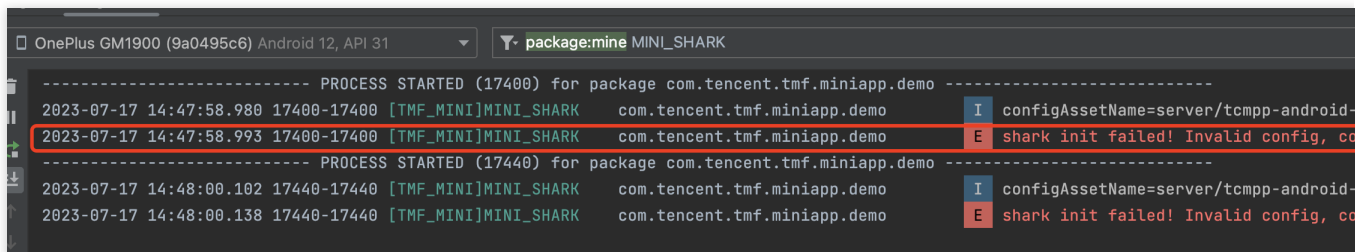
如何判断小程序 SDK 是否初始化成功呢？

从小程序 SDK mini_core 的 1.5.1.1版本之后，可通过 MINI_SHARK 过滤日志，确定初始化是否成功，初始化成功会输出：‘shark init ok!’，如图：



The screenshot shows the Android Studio interface with a log window. The top bar indicates the device is 'OnePlus GM1900 (9a0495c6) Android 12, API 31' and the package is 'package:mine MINI_SHARK'. The log output shows a JSON configuration object for the shark module, including fields like 'appKey', 'url', 'shark' (with sub-fields 'appKey', 'asymEnc', 'httpUrl', 'symEnc', 'tcpHost', 'tcpPort', 'feedback'), and 'feedback'. At the bottom, a log entry is highlighted with a red box: '2023-07-17 14:43:56.173 15685-15685 [TMF_MINI]MINI_SHARK com.tencent.tmf.miniapp.demo D shark init ok!'.

如果小程序 SDK 初始化失败，会看到如下的异常日志输出：



The screenshot shows the Android Studio interface with a log window. The top bar indicates the device is 'OnePlus GM1900 (9a0495c6) Android 12, API 31' and the package is 'package:mine MINI_SHARK'. The log output shows several lines of logs. A red box highlights a log entry: '2023-07-17 14:47:58.993 17400-17400 [TMF_MINI]MINI_SHARK com.tencent.tmf.miniapp.demo E shark init failed! Invalid config, cc'. Other logs show 'PROCESS STARTED' and 'configAssetName=server/tcmpp-android-'.

小程序 SDK 支持那些类型的设备进行调试呢？

小程序 SDK 支持 CPU 架构类型为 arm 的模拟器和真机设备。不支持x86类型的模拟器。

Android API

小程序管理 API

打开小程序

最近更新时间：2024-06-27 11:09:25

小程序管理 API 提供小程序搜索、小程序打开、小程序关闭相关的接入说明。

打开小程序

打开小程序时，会先判断本地是否有缓存的小程序，如果没有，则会自动从远程服务器上下载小程序，然后打开。如果有缓存的小程序，则会先打开本地小程序，同时后台校验服务器端是否有新版本。

说明：

如果有新版本，则下载新版小程序，下次打开时，就会使用新版小程序。

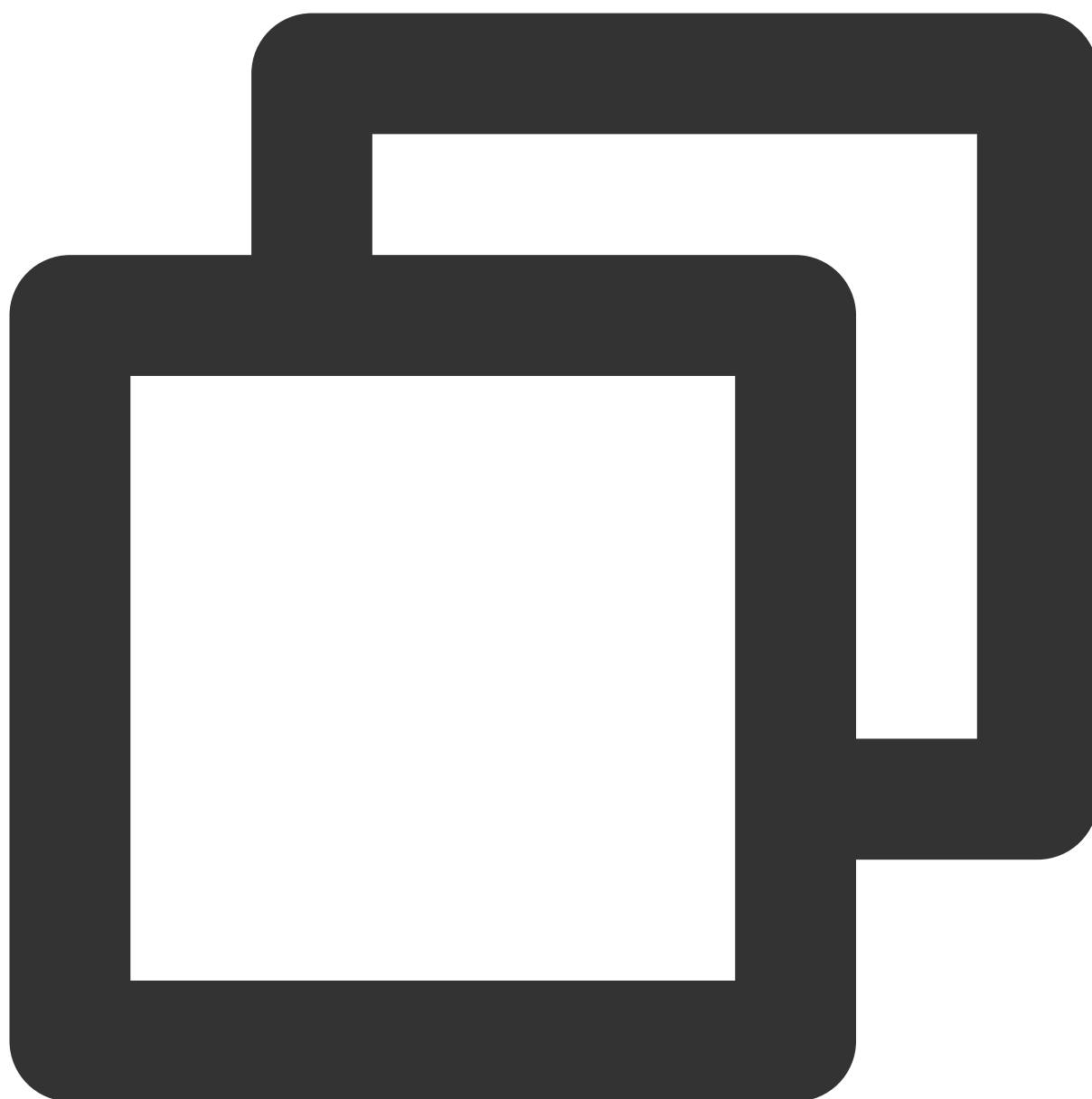
1. 根据小程序 ID（appId）打开小程序

打开正式版本（通过审核并已经发布上线）的小程序。参考如下代码：

说明：

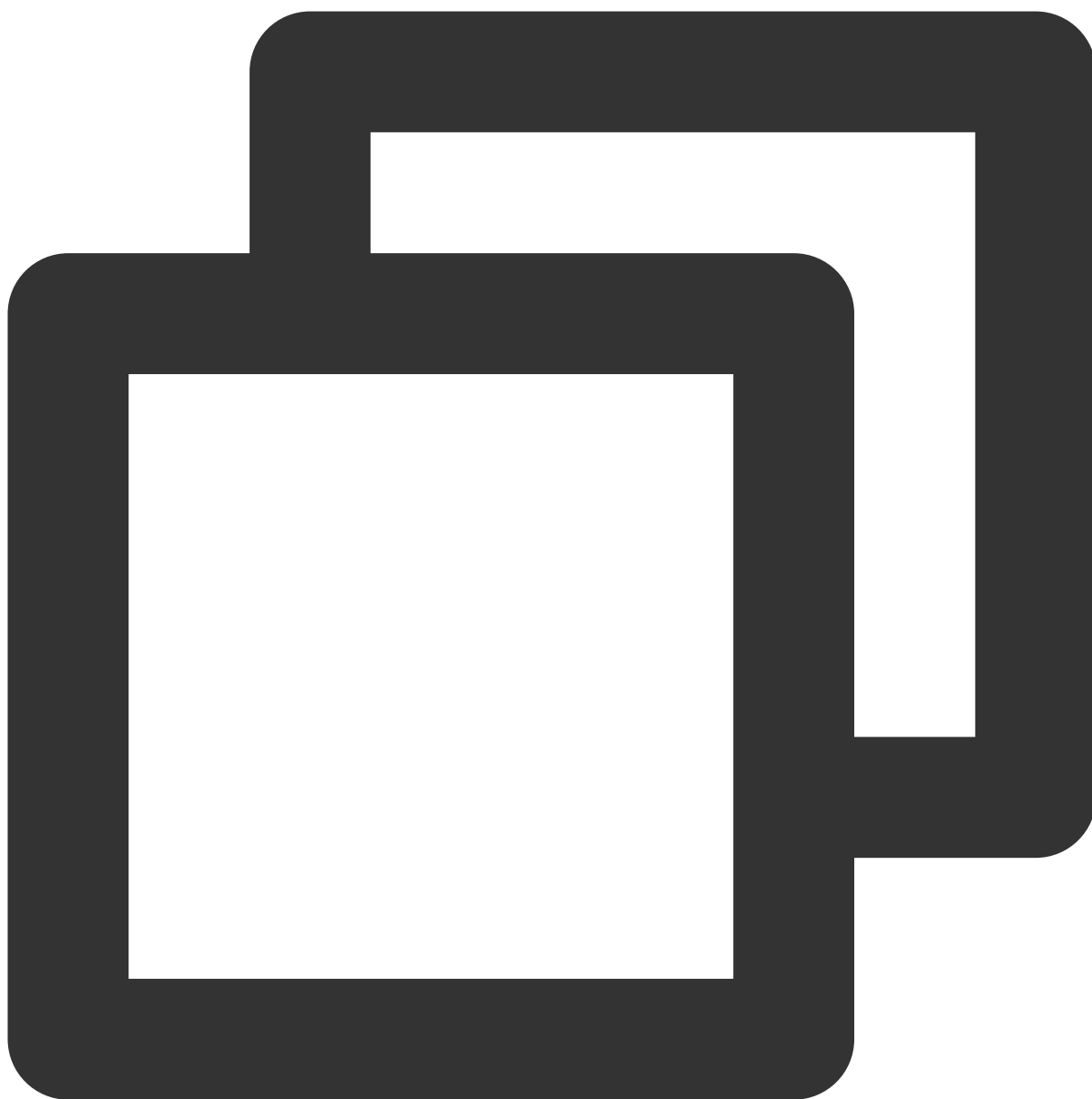
appId 字段为小程序的 ID，需要通过小程序开发者或者小程序搜索接口获取。

miniStartOptions 为小程序启动参数配置，参考 [API 说明](#)。



```
TmfMiniSDK.startMiniApp(activity, appId, miniStartOptions);
```

打开预览版或开发版小程序，参考如下代码：



```
TmfMiniSDK.startMiniApp(activiy, appId, MiniScene.LAUNCH_SCENE_MAIN_ENTRY, appVerTy
```

注意：

appVerType 应当和小程序的版本匹配；一般可以通过接口（getRecentList）返回的 MiniApp 对象实例获取 appVerType 的值。

预览版小程序的 appVerType 参数值应当为 MiniApp.TYPE_PREVIEW。

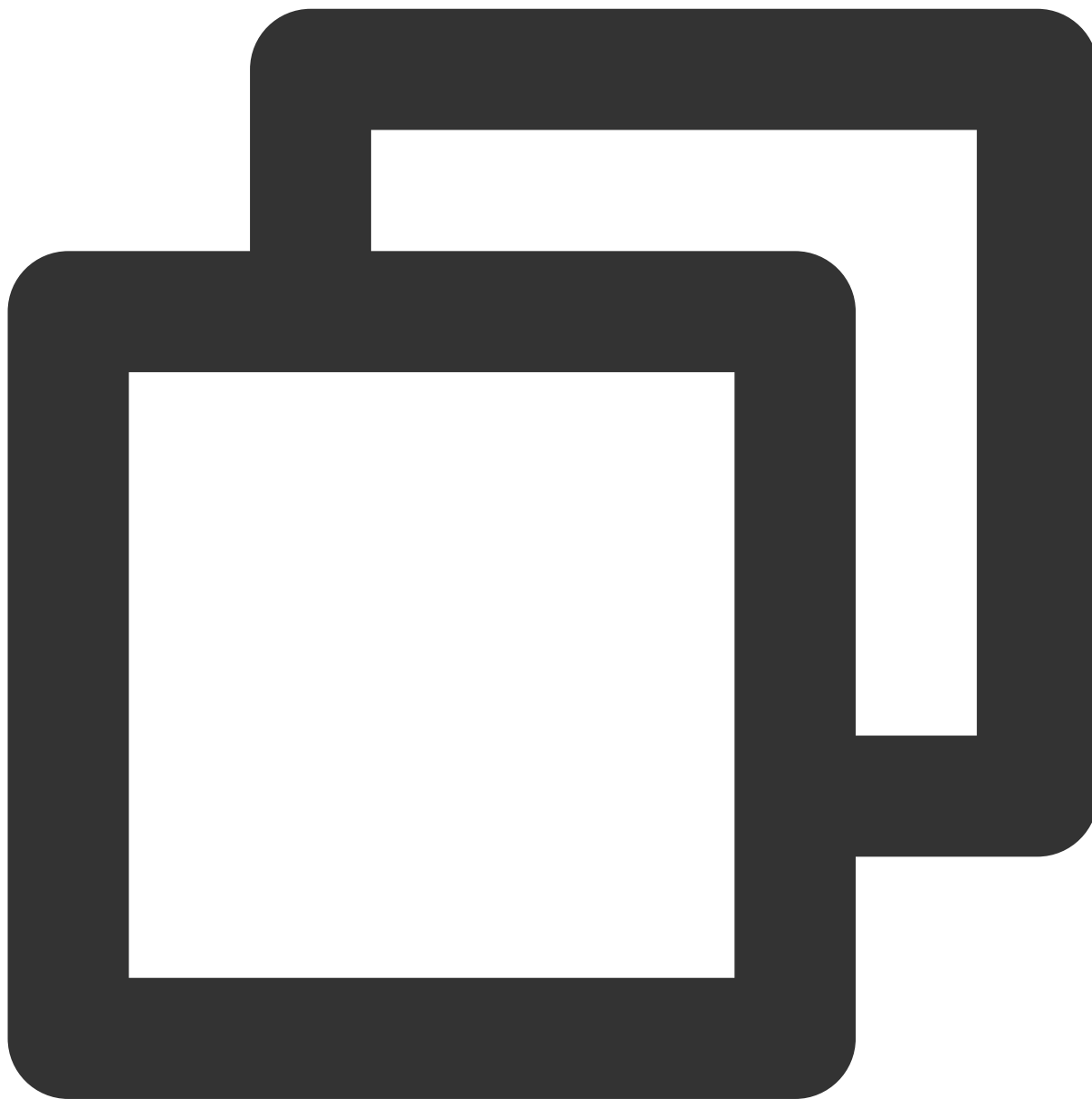
开发版小程序的 appVerType 参数值应当为 MiniApp.TYPE_DEVELOP。

2. 根据二维码内容打开小程序

小程序 SDK 提供根据平台二维码内容打开小程序的接口；在使用小程序 SDK 提供的扫码能力前，需要提前接入扫一扫的扩展能力，扫码能力的接入可以参考：扫码能力支持。

接入扫码能力后，可以通过如下的方法启动二维码扫描并打开小程序：

启动二维码扫描：

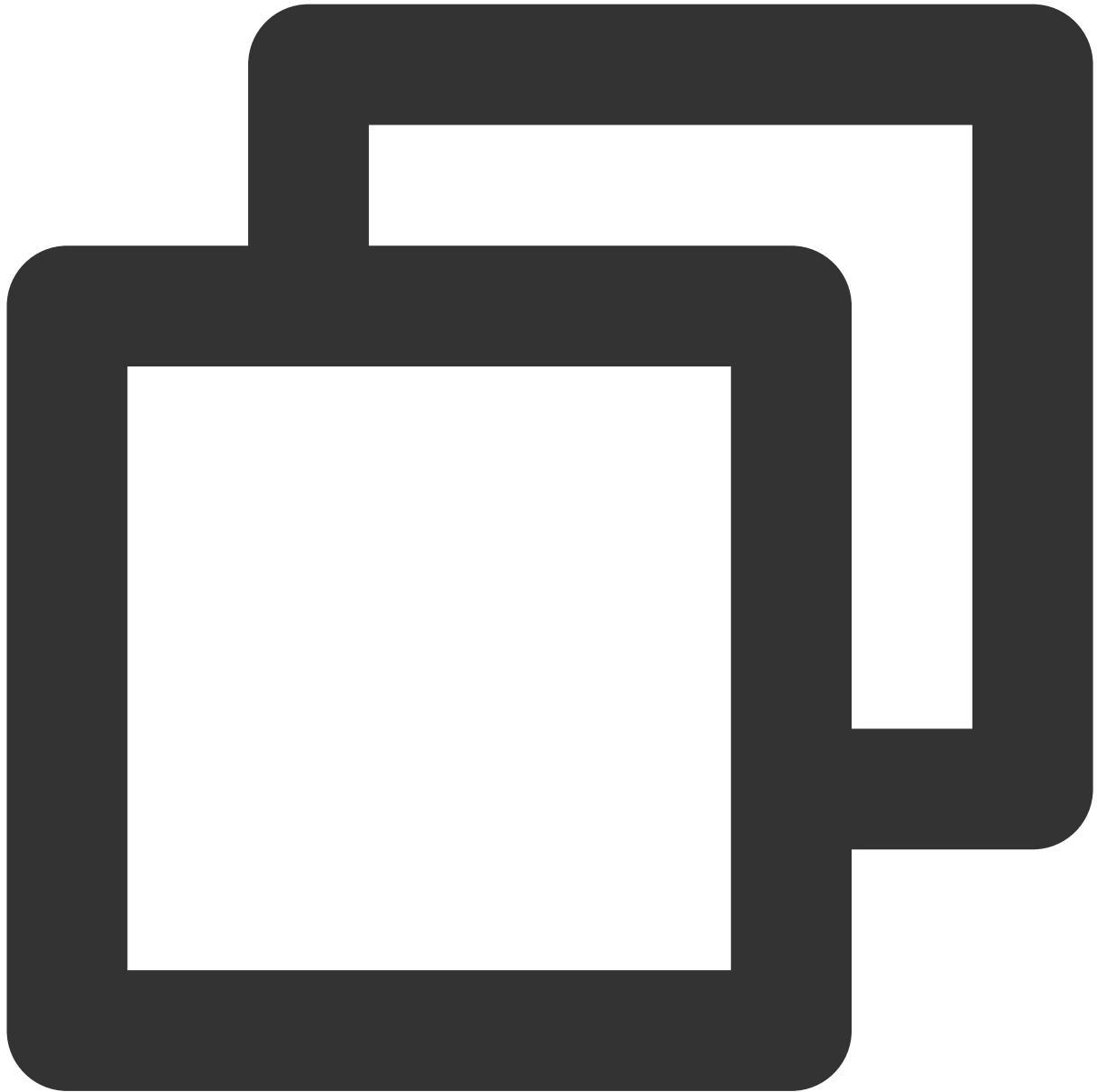


```
TmfMiniSDK.scan(activity);
```

获取二维码扫描结果：

说明：

二维码的扫描结果通过 `Activity` 的 `onActivityResult` 方法返回；为了识别二维码的内容，需要重写 `Activity` 的 `onActivityResult` 方法，并调用 `TmfMiniSDK.getScanResult` 获取二维码内容。

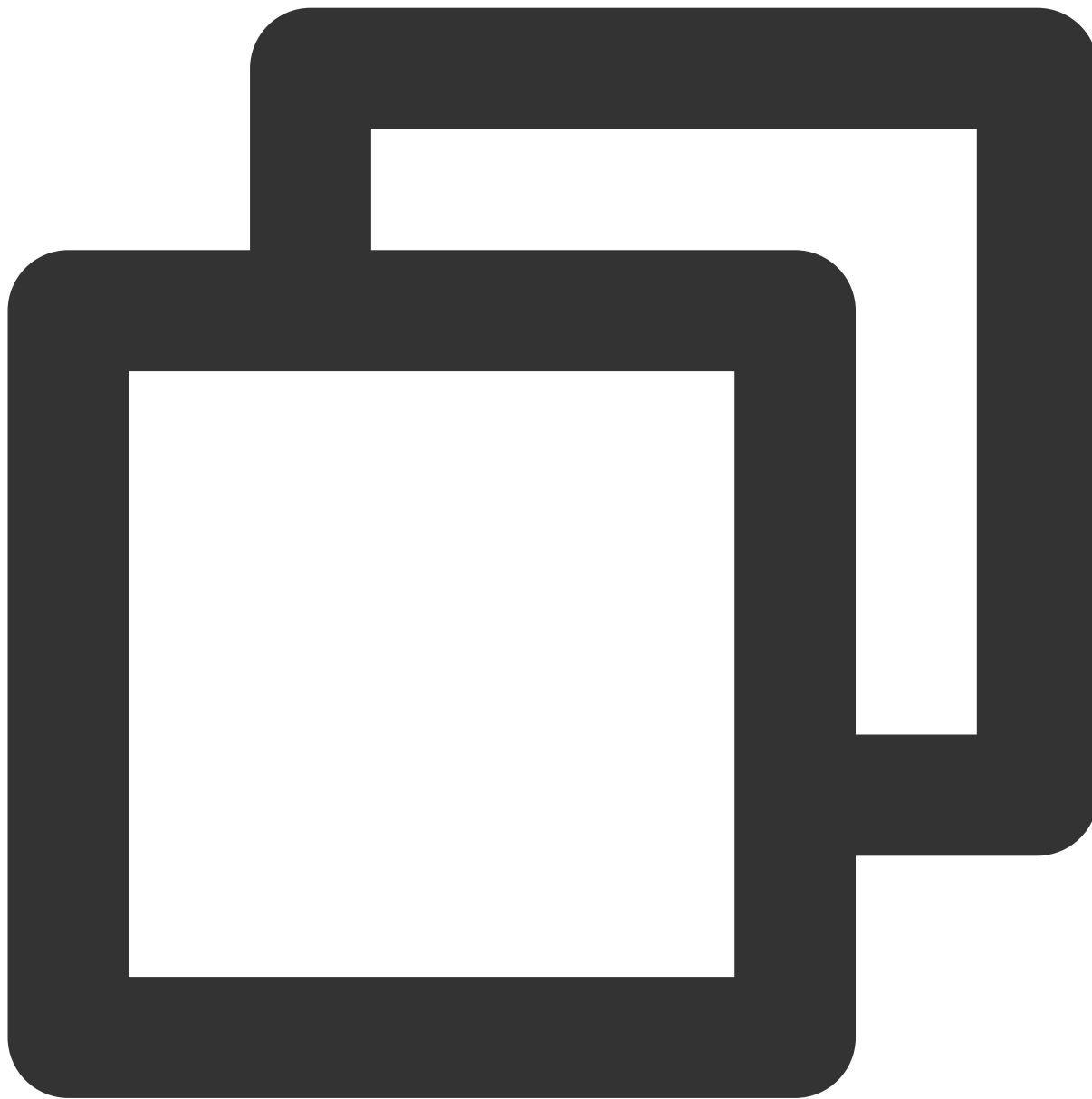


```
@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Integer data) {
    super.onActivityResult(requestCode, resultCode, data);
    //获取二维码结果
    JSONObject scanResult = TmfMiniSDK.getScanResult(requestCode, data);
}
```

根据二维码扫描结果打开小程序：

说明：

调用 `getScanResult` 方法获取的是一个 JSON 结构的数据，小程序的启动参数存放在这个 JSON 数据结构的 `'result'` 字段中，需要提取 `result` 字段所对应的值，最终将使用这个值用于打开小程序。



```
@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    //获取二维码结果
    JSONObject scanResult = TmfMiniSDK.getScanResult(requestCode, data);
    if (scanResult != null) {
        //提取result字段
    }
}
```

```
String result = scanResult.optString("result");
if (!TextUtils.isEmpty(result)) {
    MiniStartLinkOptions options = new MiniStartLinkOptions();
    //利用result字段的值打开小程序
    TmfMiniSDK.startMiniAppByLink(this, result, options);
}
}
```

3. 搜索并打开小程序

调用小程序搜索接口：

说明：

SearchOptions 的参数分别对应如下：

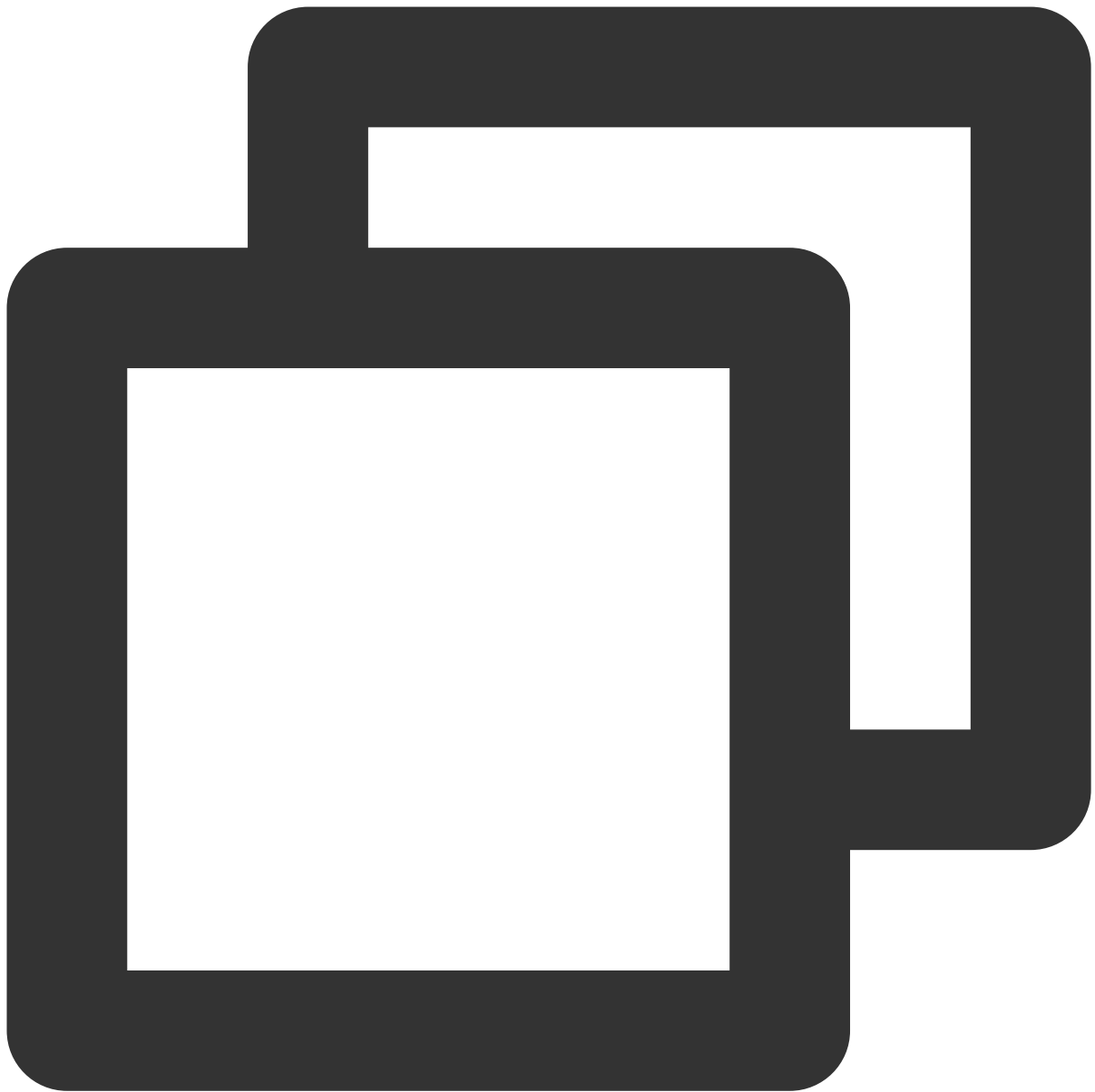
keyword：小程序关键字；

firstCate：小程序一级分类；

sencondaryCate：小程序二级分类。

小程序一级分类和二级分类同时填写，搜索结果将一级分类的结果和二级分类的结果取交集；

单独填写一级分类或者二级分类，则按照单个分类关键字进行搜索并返回结果。



```
SearchOptions searchOptions = new SearchOptions(keyword, firstCate, sencondaryCate)

TmfMiniSDK.searchMiniApp(searchOptions, new MiniCallback<List<MiniApp>>() {
    @Override
    public void value(int code, String msg, List<MiniApp> data) {
        if (code == MiniCode.CODE_OK && data != null) {
            //搜索成功，小程序列表不为空
        } else {
            //搜索失败或者小程序列表为空
        }
    }
})
```

```
}  
});
```

打开搜索结果中的小程序：

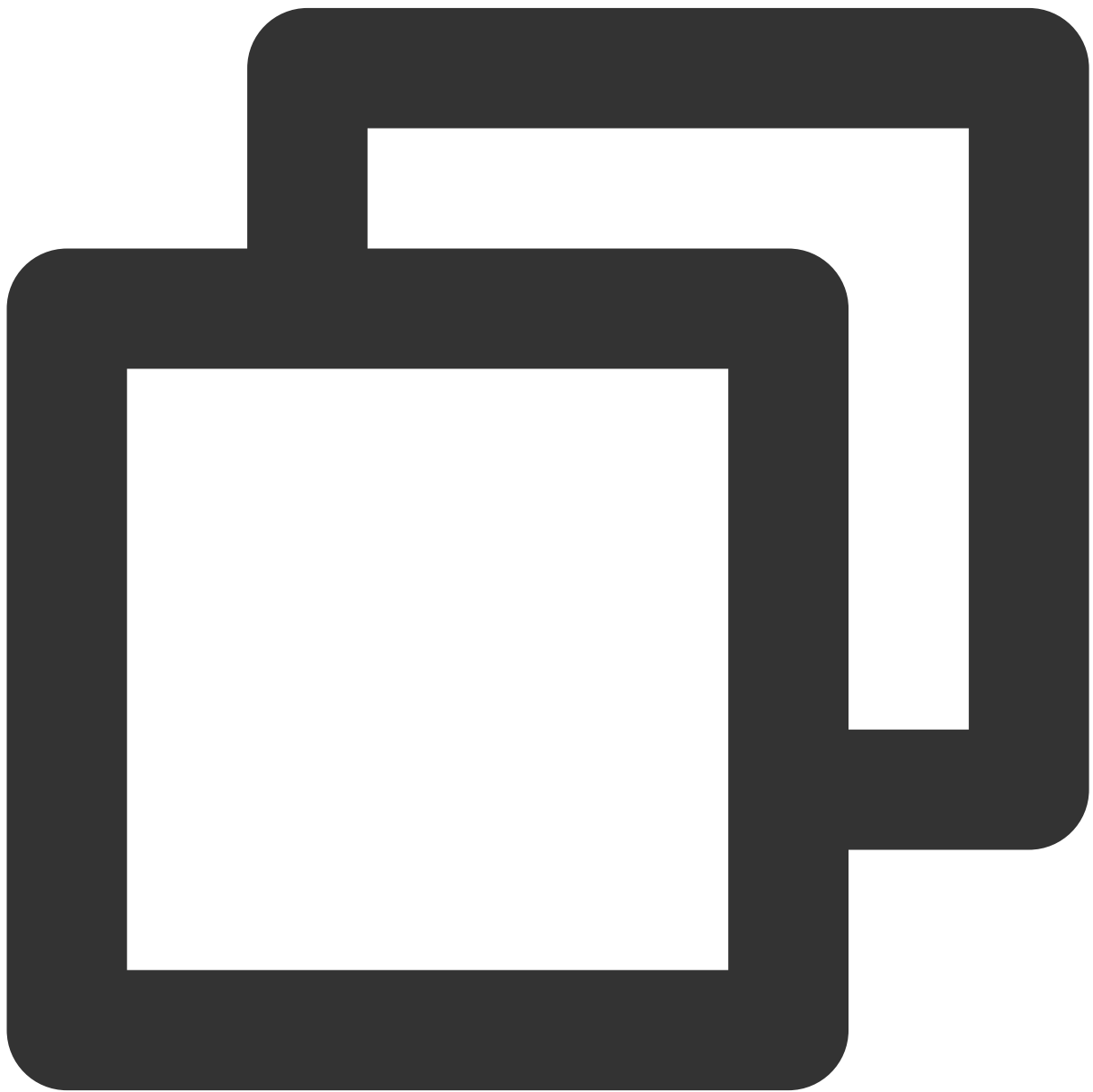
说明：

小程序的搜索结果通过 `MiniCallback` 接口的 `value` 方法进行返回。

参数 `int code` 表示错误码。

参数 `String msg` 表示搜索返回信息。

参数 `List<MiniApp> data` 表示搜索到的小程序信息列表。



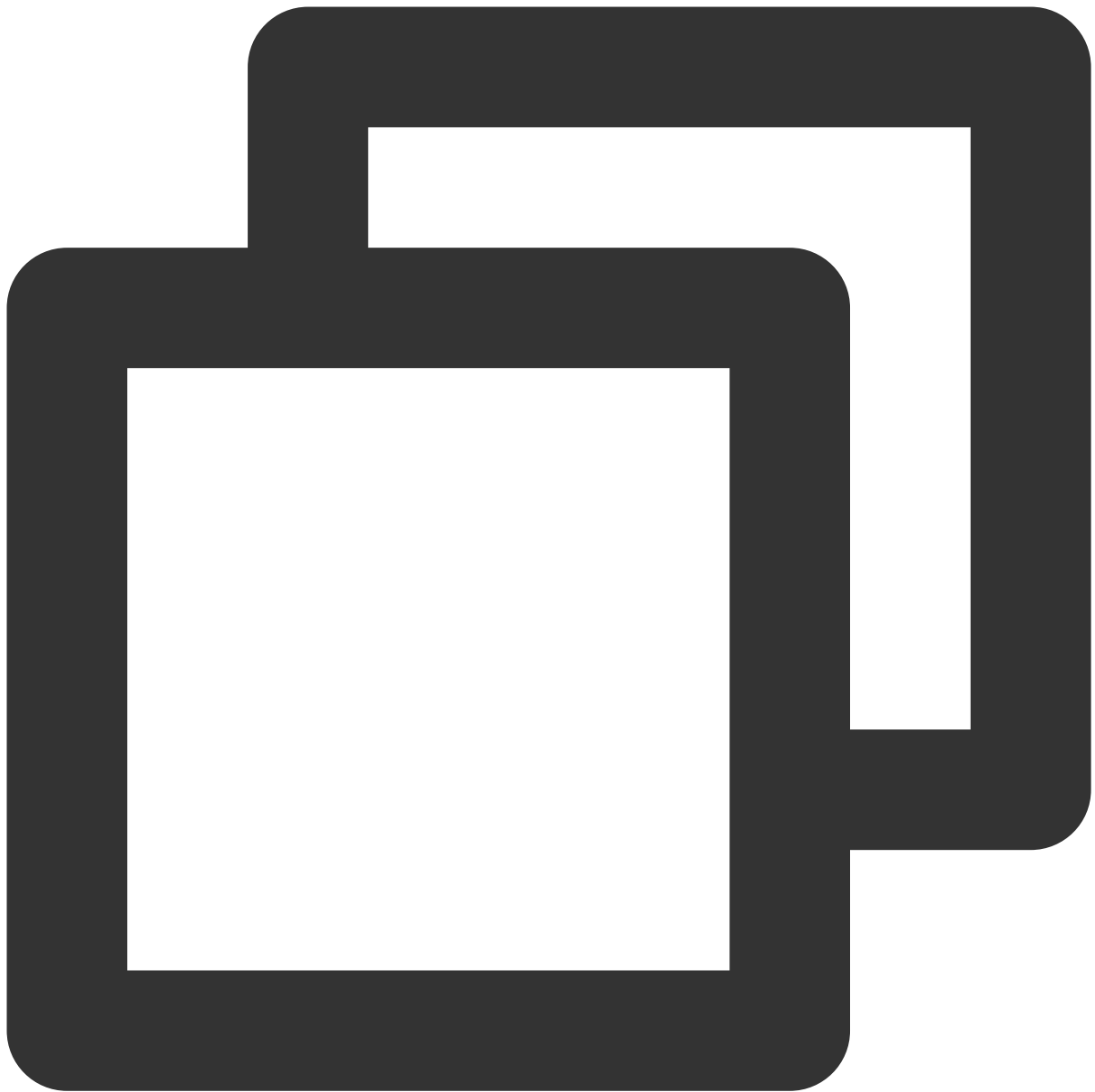
```
SearchOptions searchOptions = new SearchOptions(keyword, firstCate, sencondaryCate)

TmfMiniSDK.searchMiniApp(searchOptions, new MiniCallback<List<MiniApp>>() {
    @Override
    public void value(int code, String msg, List<MiniApp> data) {
        if (code == MiniCode.CODE_OK && data != null) {
            //搜索成功，小程序列表不为空
            MiniApp miniApp = data.get(0);
            //打开搜索结果中的第一个小程序
            TmfMiniSDK.startMiniApp(this, miniApp.appId, MiniScene.LAUNCH_SCENE_SEARCH
        } else {
            //搜索失败或者小程序列表为空
        }
    }
});
```

4. 小程序启动监听

为了方便开发者排查问题，小程序 SDK 提供小程序启动错误的监听，可以通过 `MiniStartOptions` 的 `resultReceiver` 进行设置，

设置方法如下：



```
private ResultReceiver mResultReceiver = new ResultReceiver(new Handler()) {
    @Override
    protected void onReceiveResult(int resultCode, Bundle resultData) {
        if (resultCode != MiniCode.CODE_OK) {
            String errMsg = resultData.getString("errMsg");
            Toast.makeText(mActivity, errMsg + resultCode, Toast.LENGTH_SHORT
        }
    }
};
```

`miniStartOptions.resultReceiver` 可用于接收小程序启动错误情况，所以返回的错误码参考 [API 说明](#)，里面都有对应的说明。

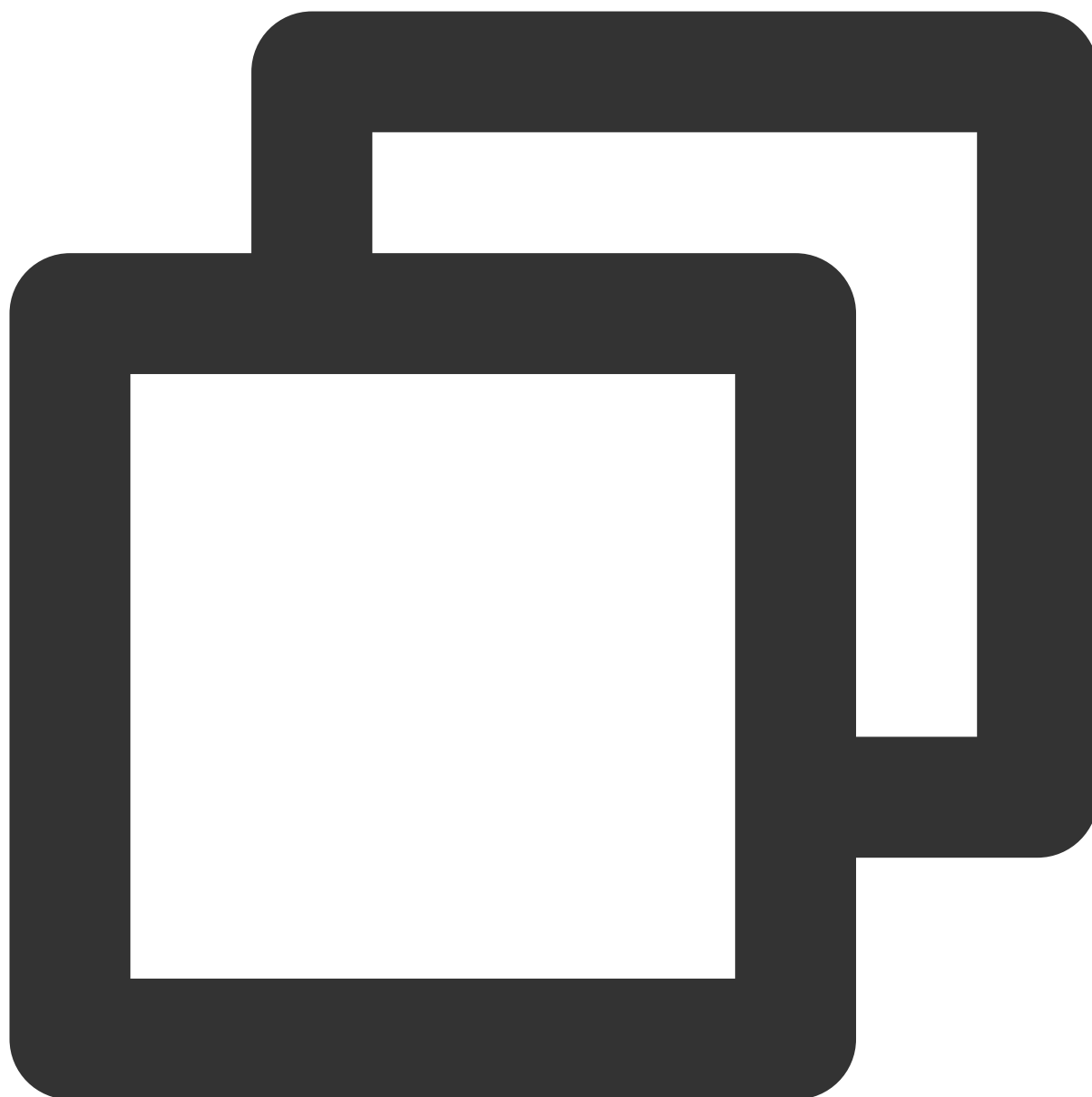
关闭小程序

最近更新时间：2024-06-27 11:09:15

调用关闭小程序接口，将终止小程序的一切活动，并终止小程序进程。

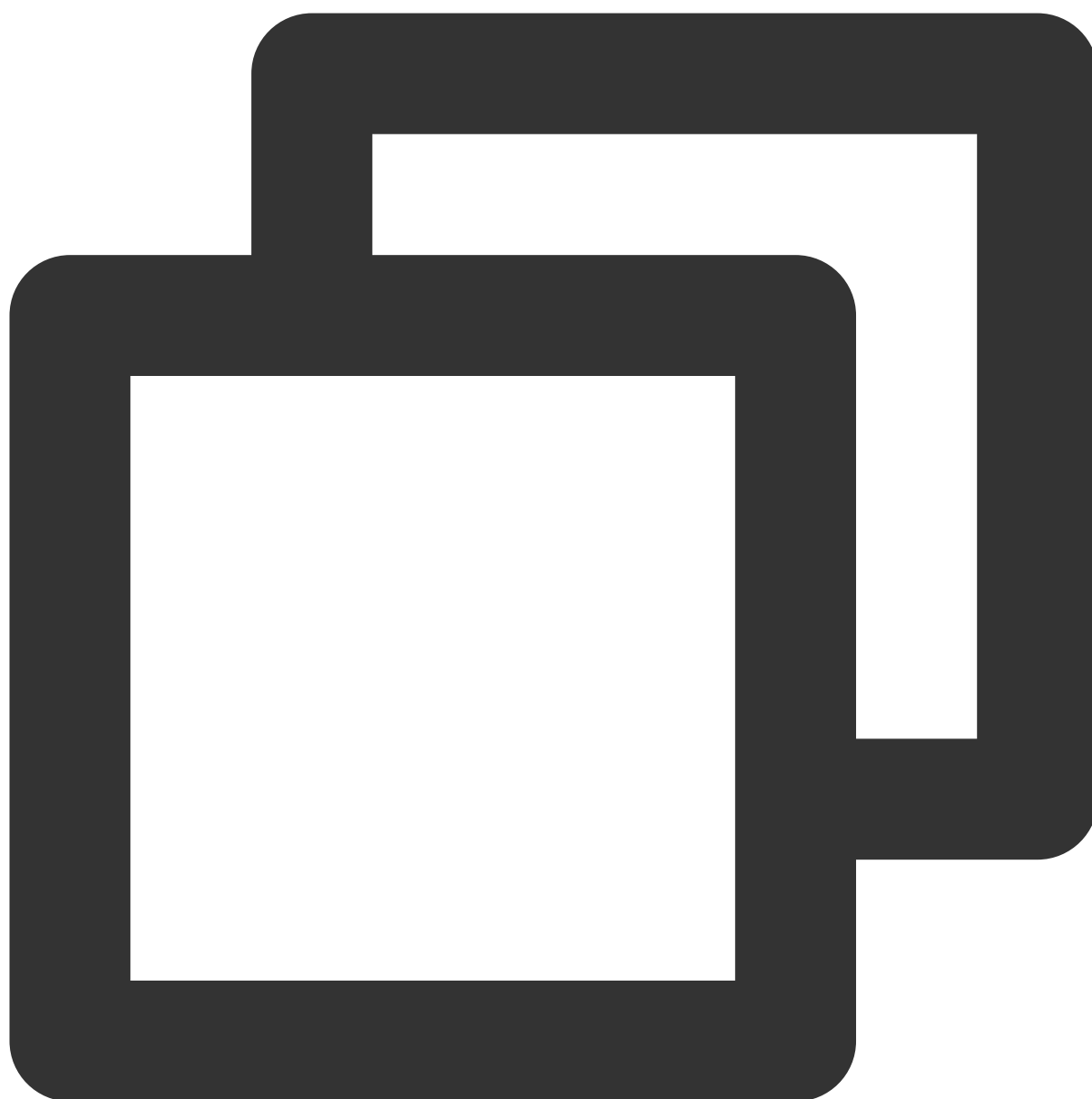
关闭指定小程序

API 说明：可以通过此接口关闭指定小程序ID的小程序。



```
/**
 * 停止小程序
 *
 * @param context
 * @param appId
 */
public static void stopMiniApp(Context context, String appId)
```

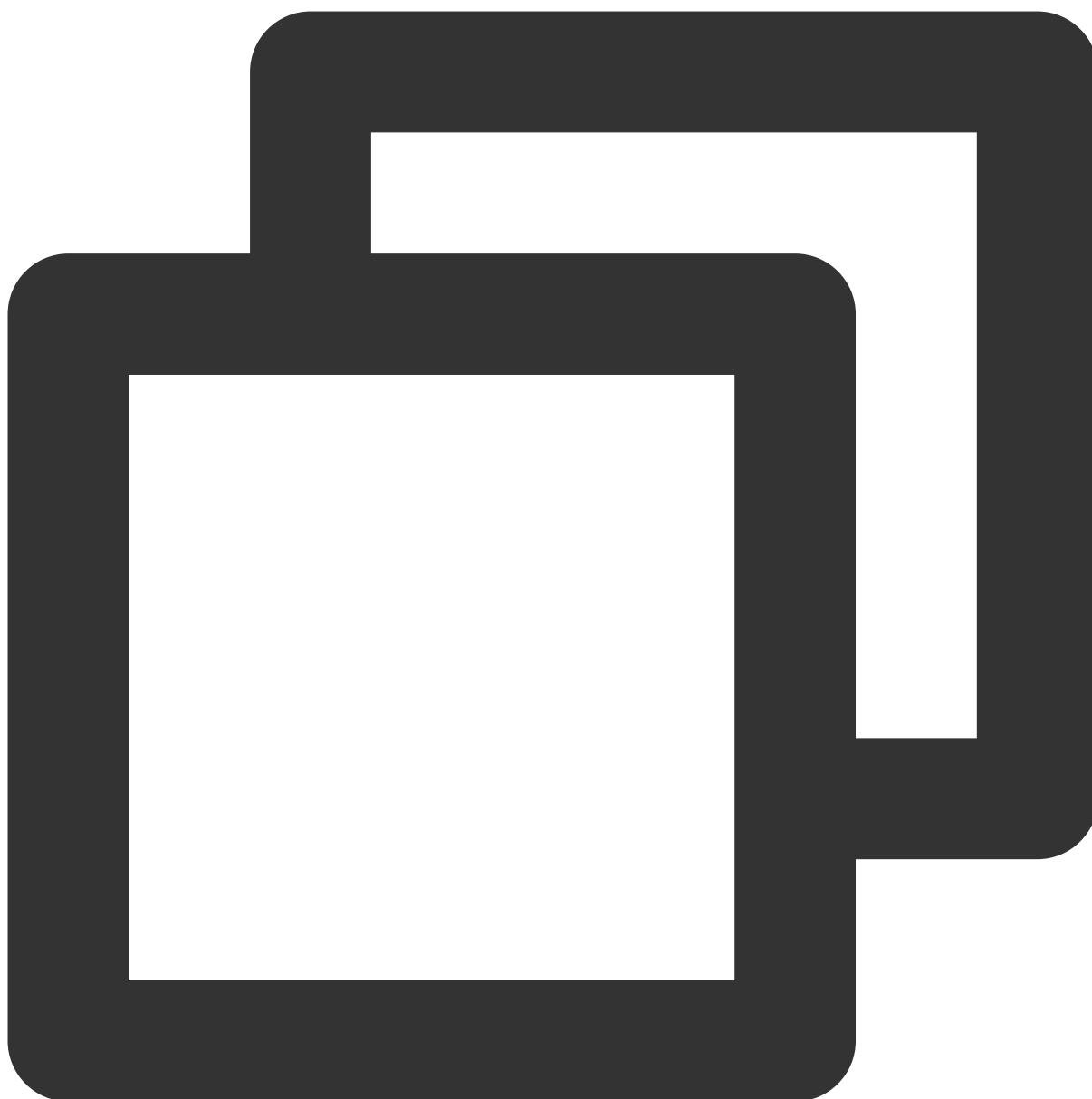
示例代码：



```
TmfMiniSdk.stopMiniApp(context, appId);
```

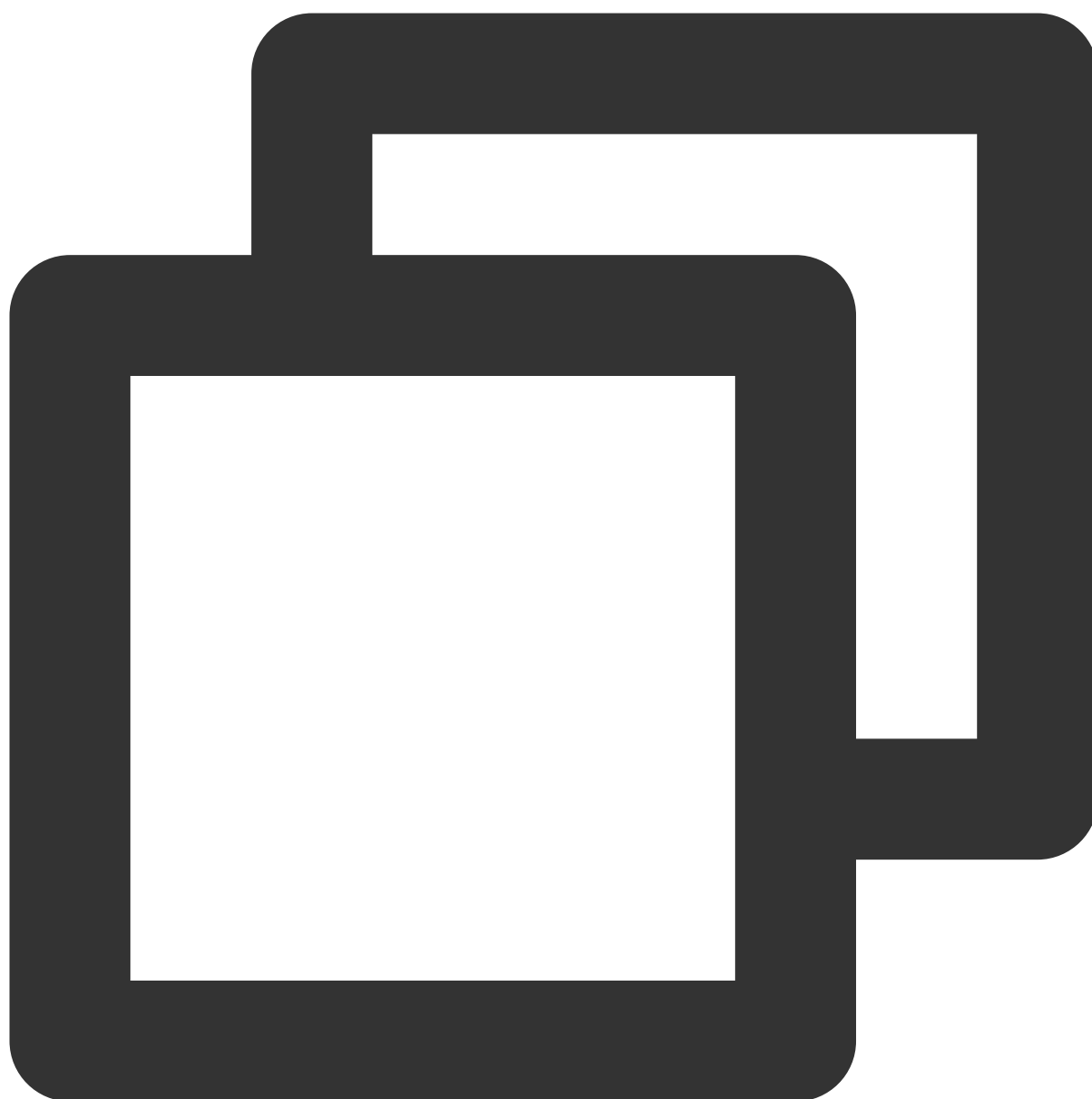
关闭所有小程序

API 说明：可以通过此接口关闭所有小程序。



```
/**
 * 停止所有小程序
 *
 * @param context 上下文
 */
public static void stopAllMiniApp(Context context)
```

示例代码：



```
TmfMiniSdk.stopAllMiniApp(context);
```

删除小程序

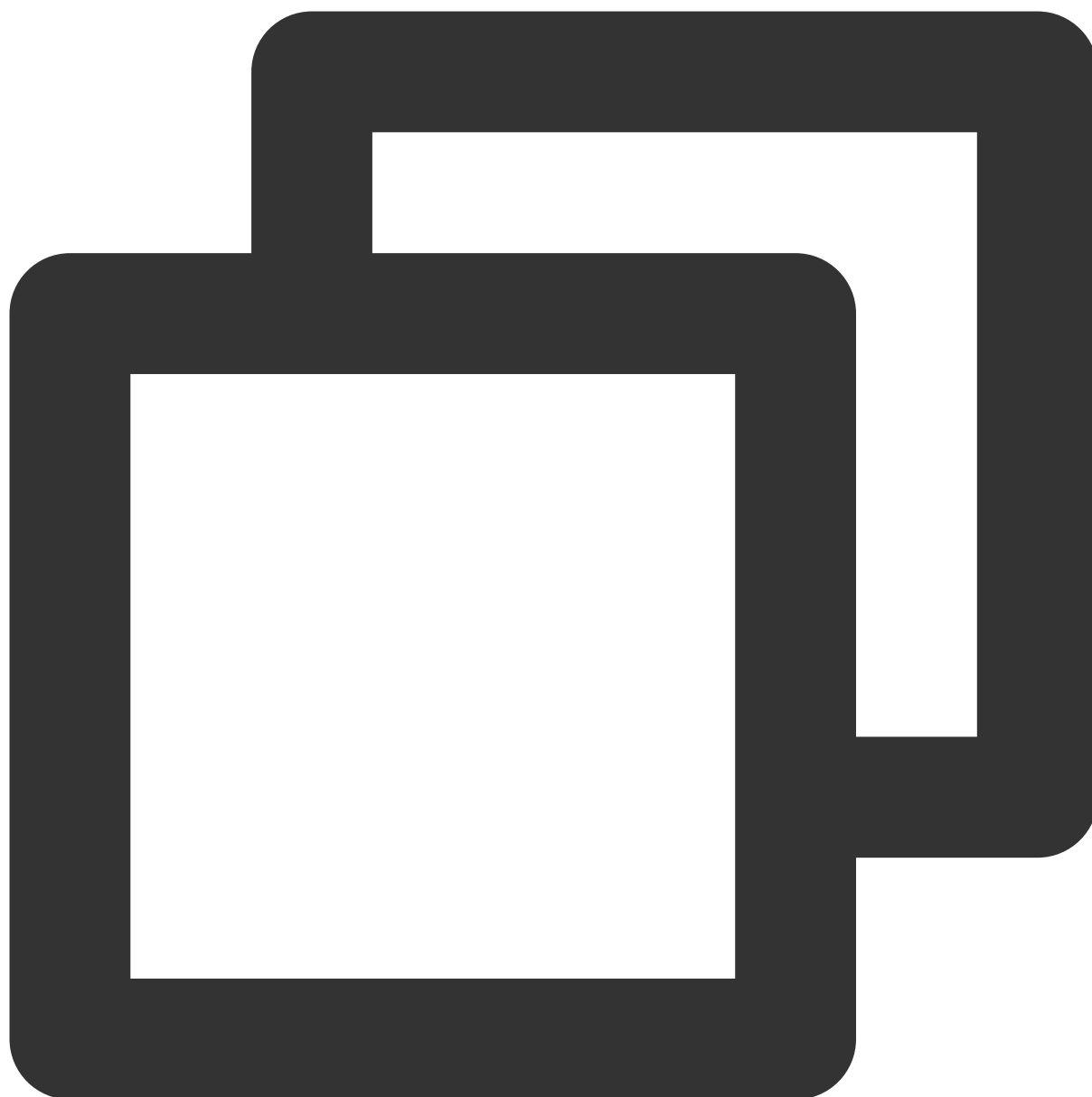
最近更新时间：2024-06-27 11:09:04

说明：

由于小程序的运行，会将小程序包和小程序信息缓存在本地，以便下次更快速的打开。如果想要将小程序的所有信息都删除，可以调用以下 **API** 删除某个小程序或者删除所有小程序。

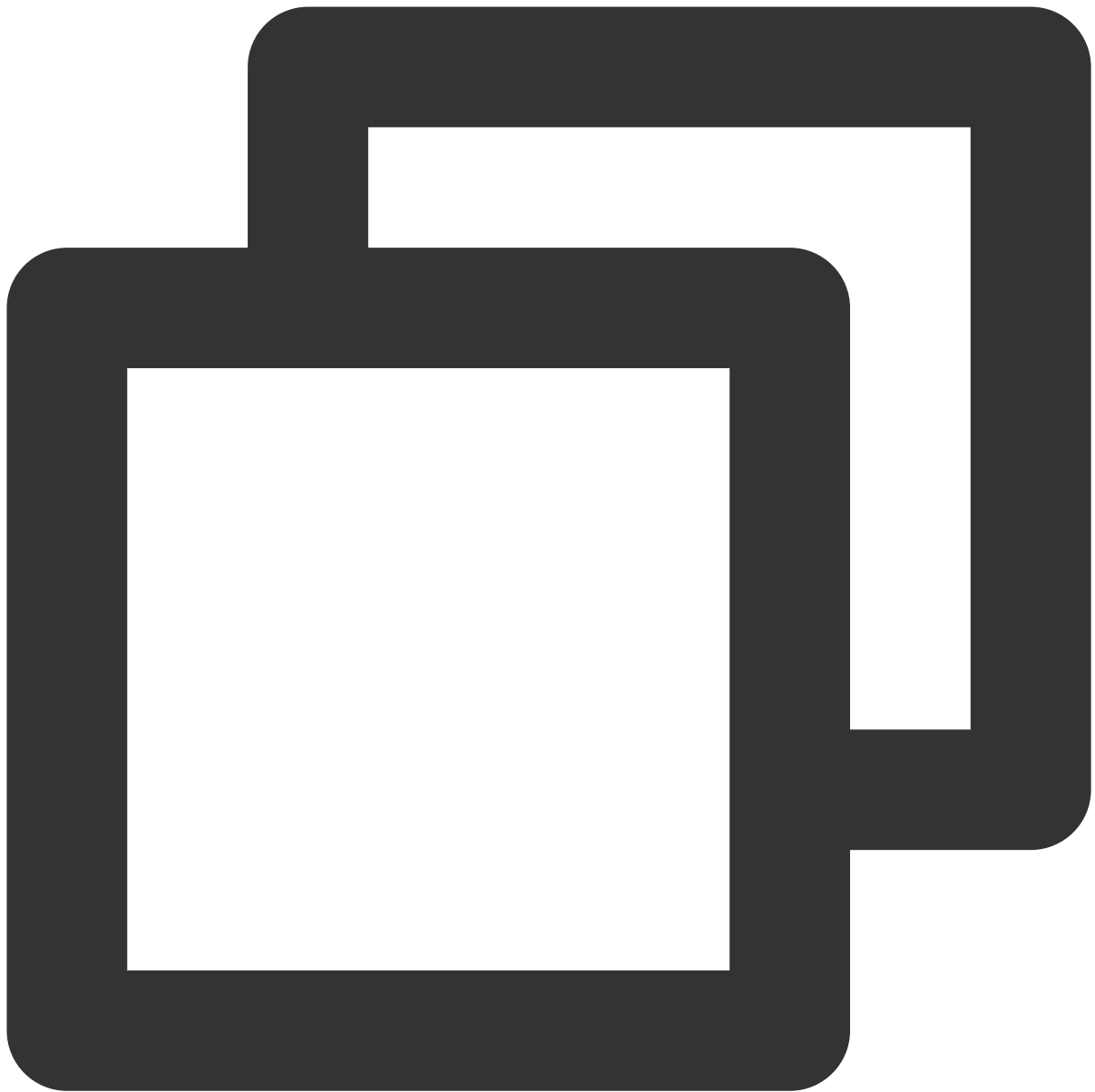
根据小程序 ID 删除

API 说明：



```
/**
 * 根据 appId 删除小程序(正式、开发、预览版都会删除)
 * @param appId
 */
public static void deleteMiniApp(String appId)
```

示例代码：

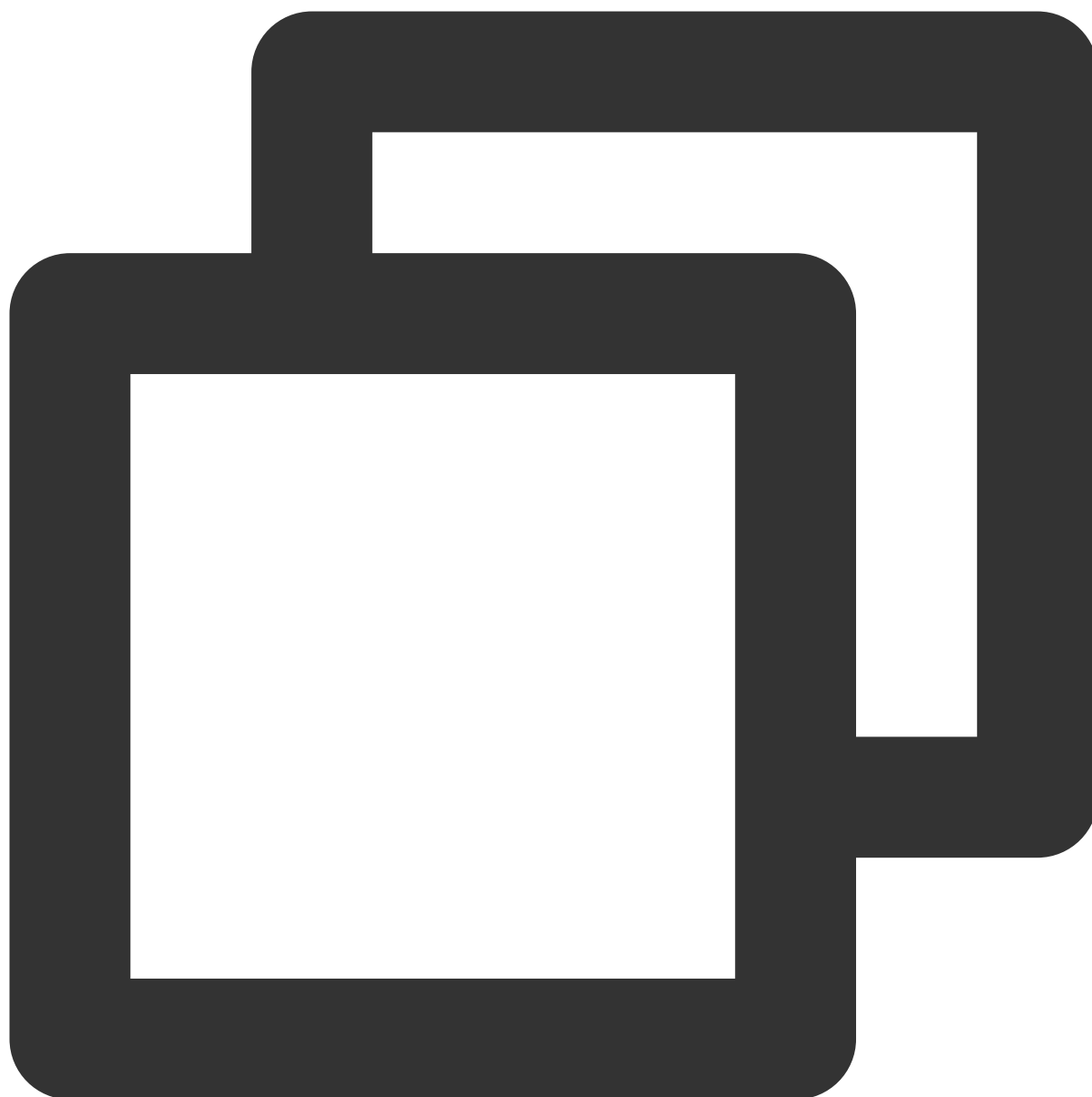


```
//删除指定小程序ID的小程序  
TmfMiniSdk.deleteMiniApp(appId);
```

删除指定版本和类型的小程序：

根据指定的版本类型和版本号删除小程序。

API 说明：



```
/**
 * 删除指定类型和版本小程序
 * @param appId
 * @param appVerType
 * @param version
 */
public static void deleteMiniApp(String appId, int appVerType, String version)
```

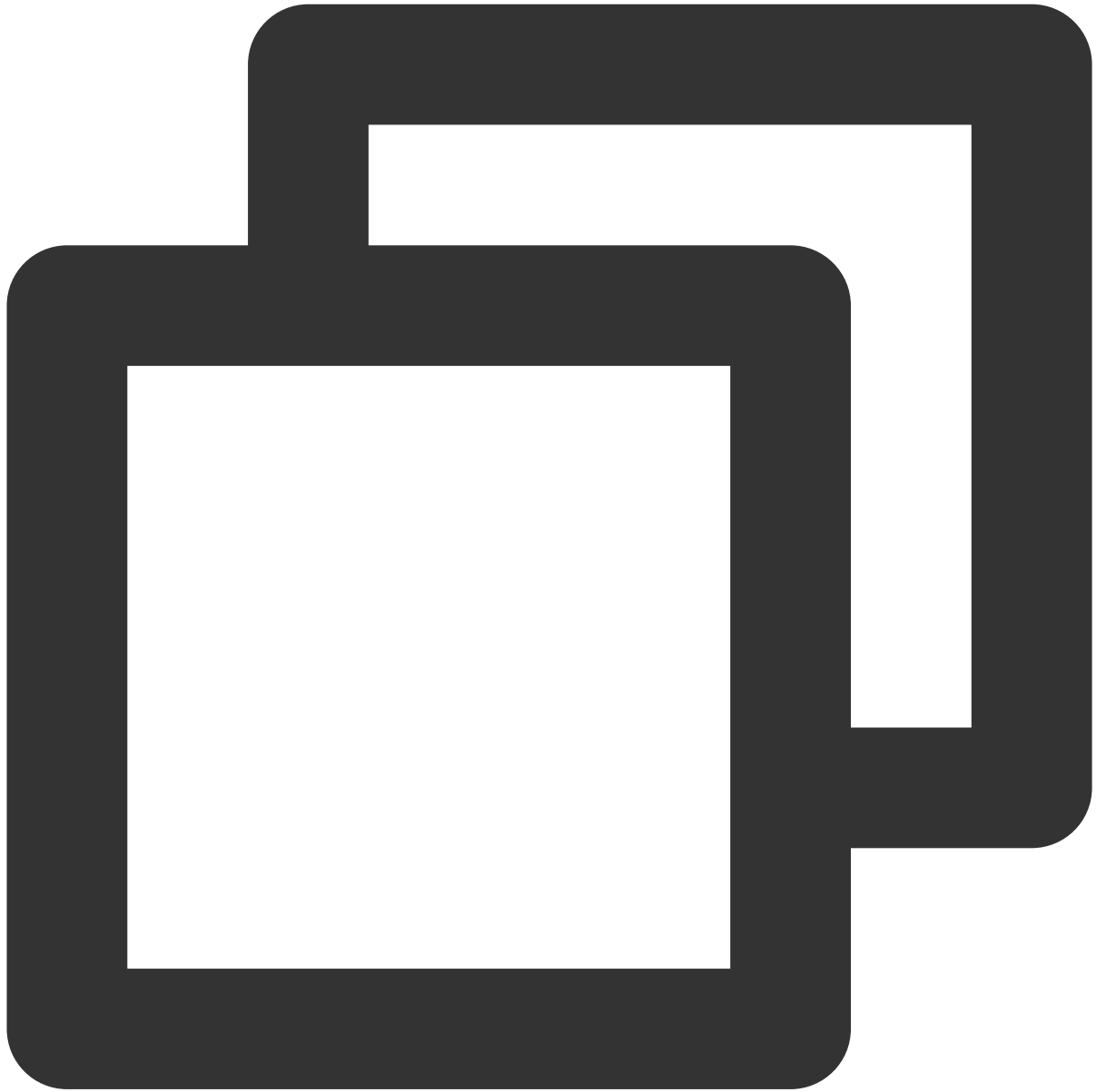
示例代码：

说明：

参数 `appId` 为小程序的 ID。

参数 `appVerType` 为小程序类型，可以参考小程序类型说明。

参数 `version` 为小程序版本号。



```
//删除指定小程序ID和类型以及版本的小程序  
TmfMiniSdk.deleteMiniApp(appId, appVerType, version);
```

搜索小程序

最近更新时间：2024-06-27 11:08:54

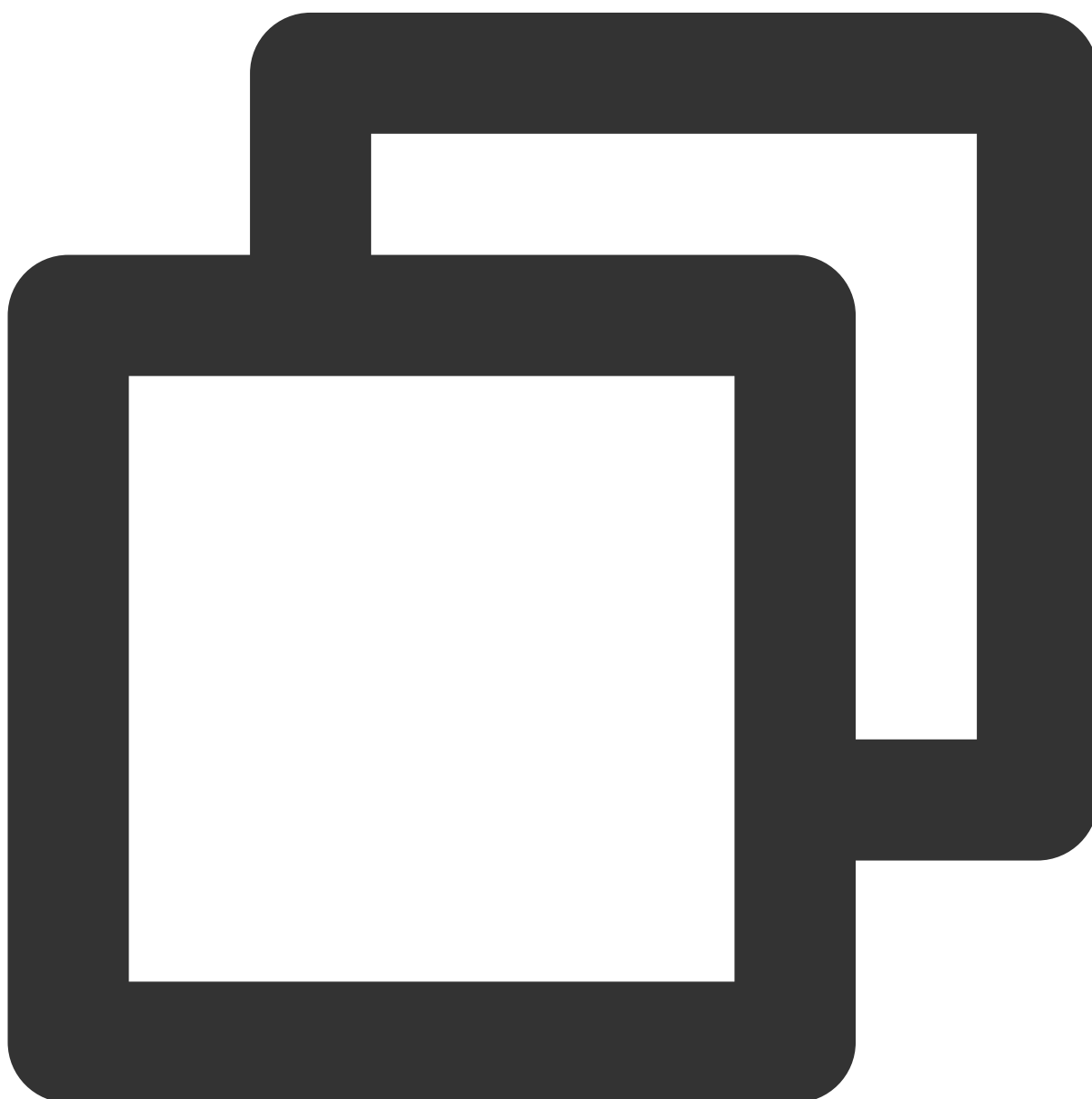
小程序 SDK 提供线上小程序搜索的接口，可以通过此接口对小程序按关键字和分类进行搜索。

API 说明：

说明：

`SearchOptions` 参数用于指定小程序搜索的关键字和分类信息。

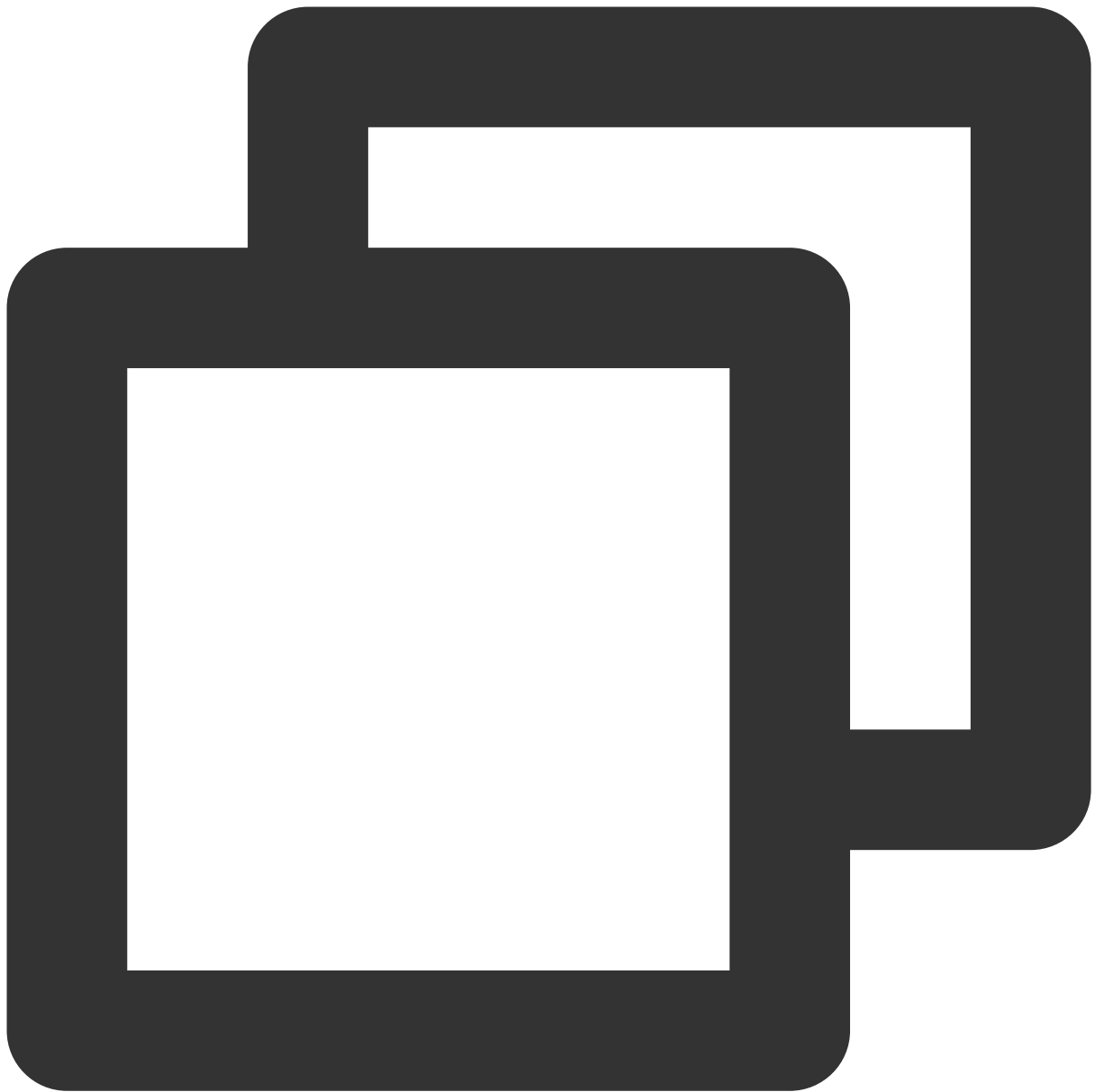
`MiniCallback` 参数用于获取小程序的搜索结果。




```
/**
 * 小程序搜索
 *
 * @param searchOptions
 * @param callback
 */
public static void searchMiniApp(SearchOptions searchOptions, MiniCallback<List<Min
```

按关键字搜索

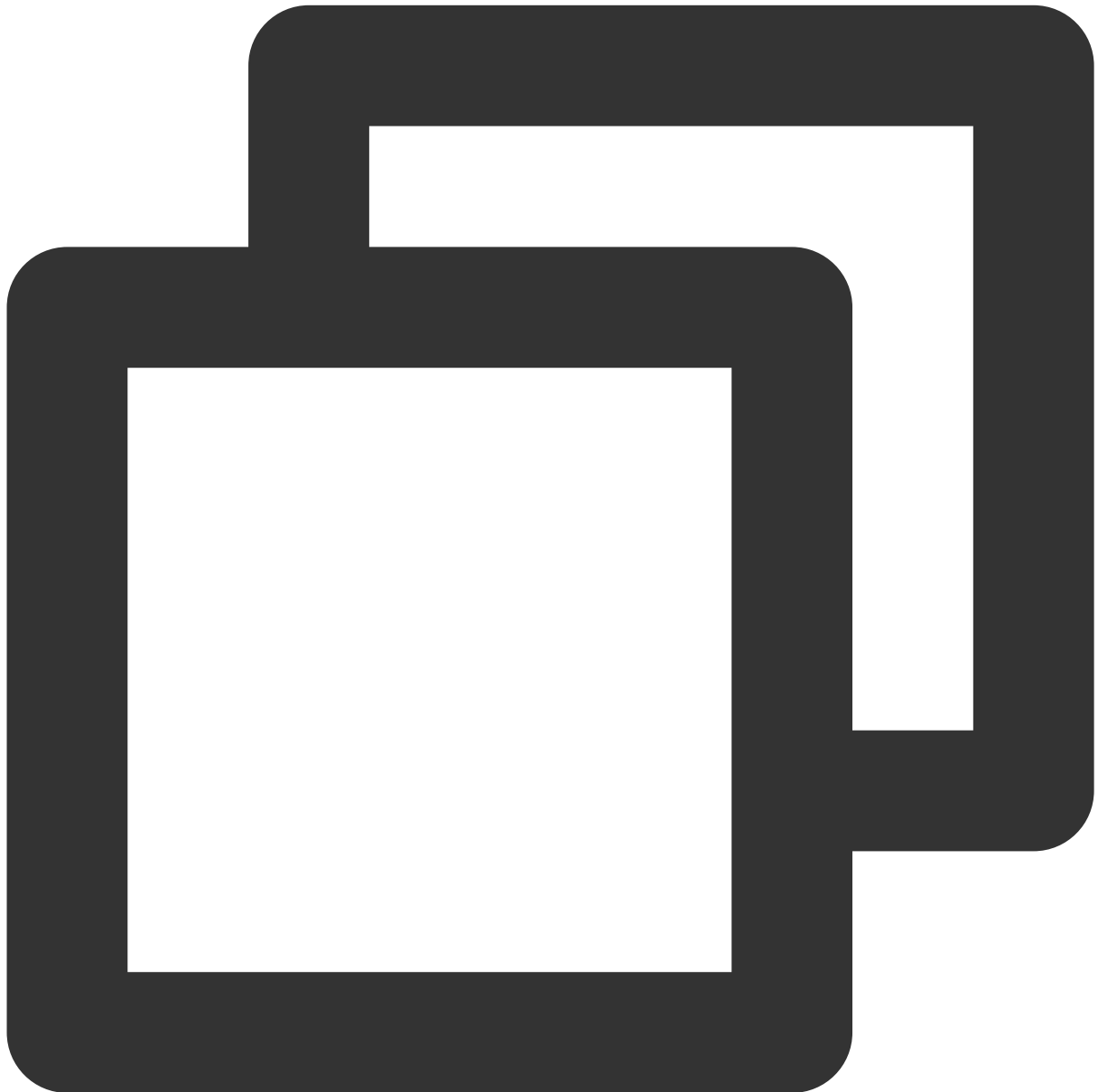
示例代码：



```
SearchOptions searchOptions = new SearchOptions("yourkeyword");
TmfMiniSDK.searchMiniApp(searchOptions, new MiniCallback<List<MiniApp>>() {
    @Override
    public void value(int code, String msg, List<MiniApp> data) {
        if (code == MiniCode.CODE_OK && data != null) {
            //搜索成功，列表不为空
        }else{
            //搜索失败，或列表为空
        }
    }
});
```

按单分类搜索

示例代码：



```
SearchOptions searchOptions = new SearchOptions("", "分类名称", "");  
TmfMiniSDK.searchMiniApp(searchOptions, new MiniCallback<List<MiniApp>>() {  
    @Override  
    public void value(int code, String msg, List<MiniApp> data) {
```

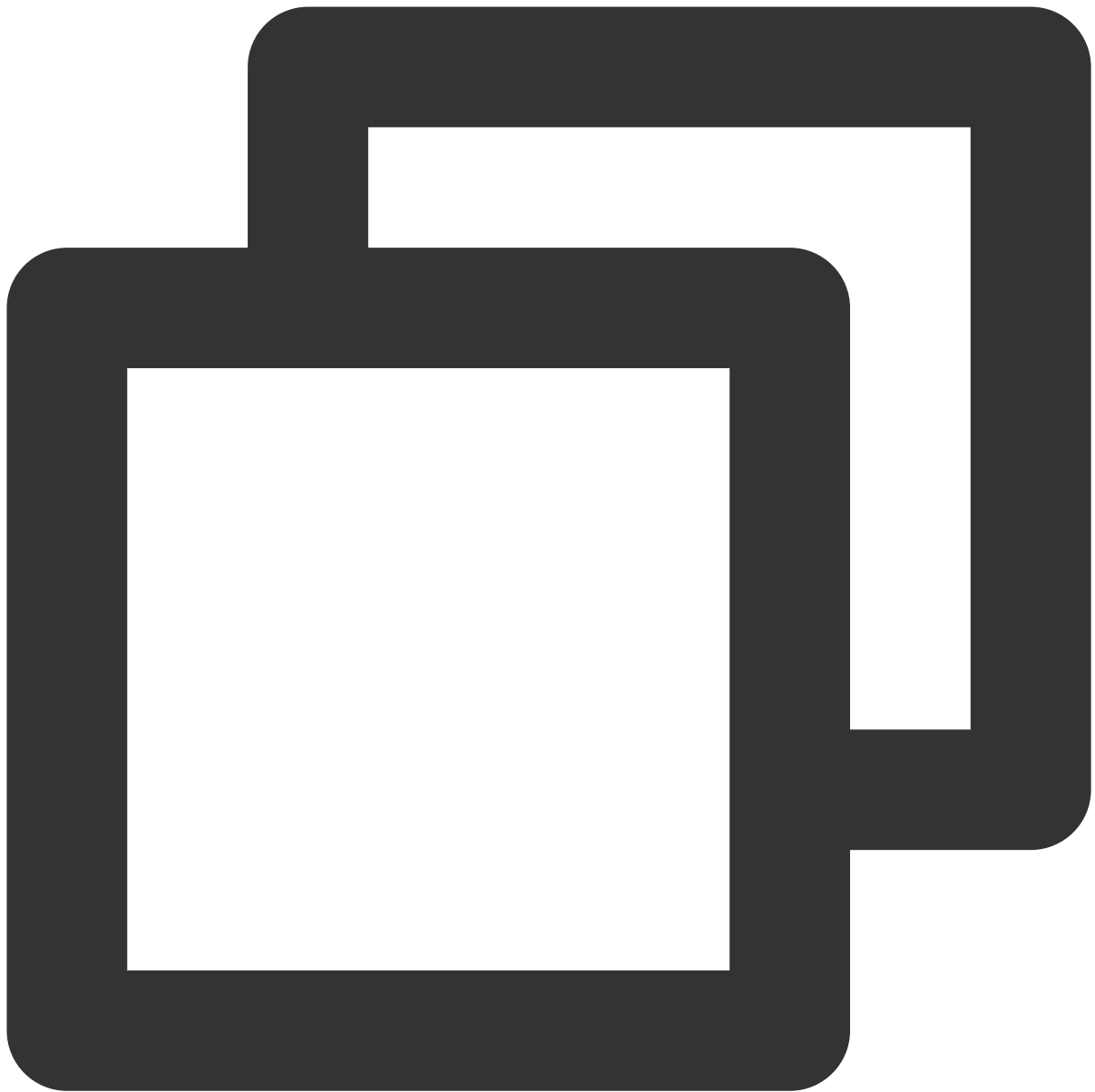
```
        if (code == MiniCode.CODE_OK && data != null) {  
            //搜索成功，列表不为空  
        }else{  
            //搜索失败，或列表为空  
        }  
    }  
});
```

双分类搜索

说明：

双分类搜索的结果为两个分类的交集。

示例代码：



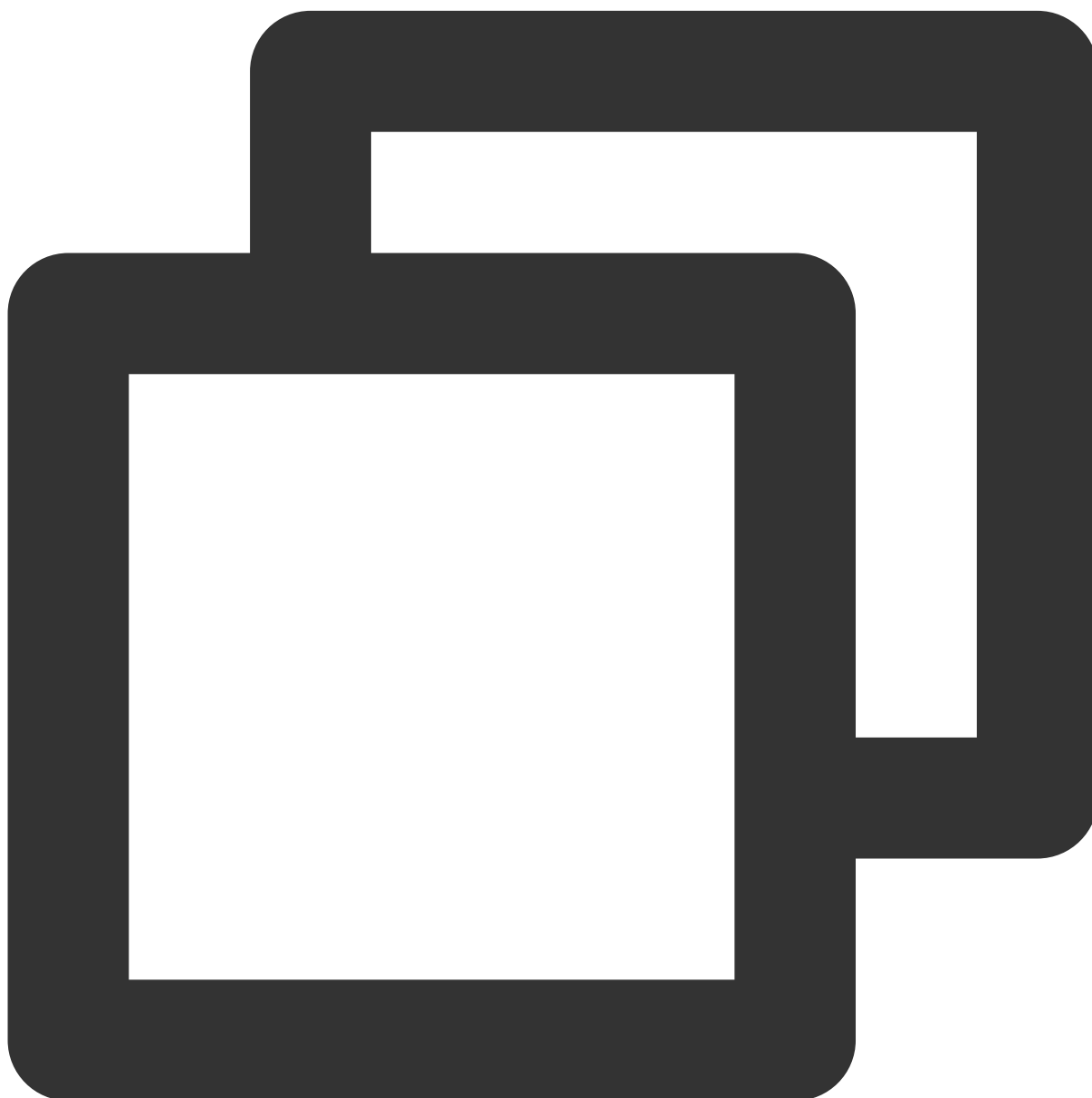
```
SearchOptions searchOptions = new SearchOptions("", "分类名称", "分类名称2");
TmfMiniSDK.searchMiniApp(searchOptions, new MiniCallback<List<MiniApp>>() {
    @Override
    public void value(int code, String msg, List<MiniApp> data) {
        if (code == MiniCode.CODE_OK && data != null) {
            //搜索成功，列表不为空
        }else{
            //搜索失败，或列表为空
        }
    }
});
```

关键字和分类搜索

说明：

搜索参数同时设置关键字分类，搜索结果为关键字和分类的交集。

示例代码：



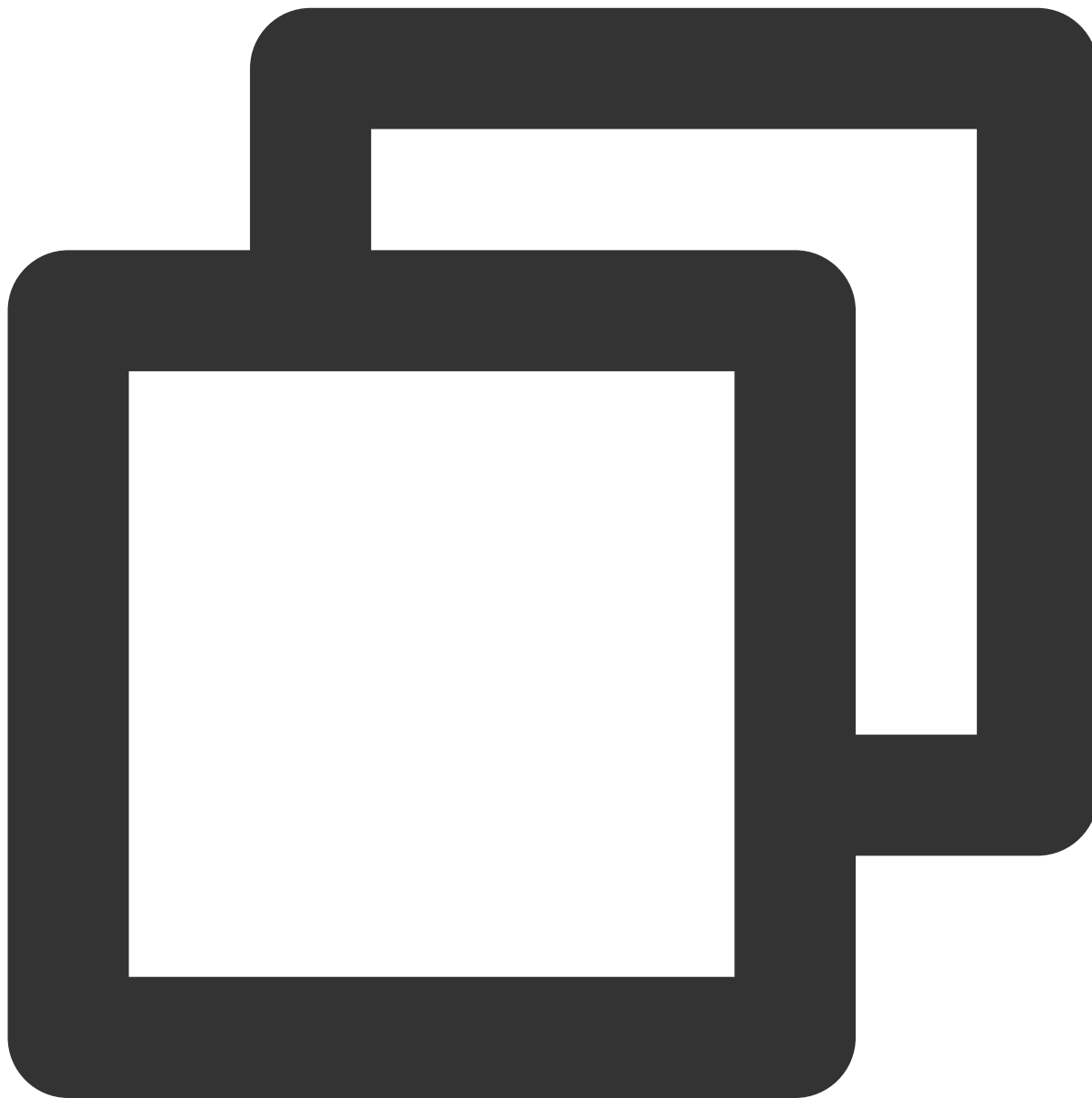
```
SearchOptions searchOptions = new SearchOptions("keyword", "分类名称", "分类名称2");  
TmfMiniSDK.searchMiniApp(searchOptions, new MiniCallback<List<MiniApp>>() {
```

```
@Override
public void value(int code, String msg, List<MiniApp> data) {
    if (code == MiniCode.CODE_OK && data != null) {
        //搜索成功，列表不为空
    }else{
        //搜索失败，或列表为空
    }
}
});
```

获取小程序最近访问列表

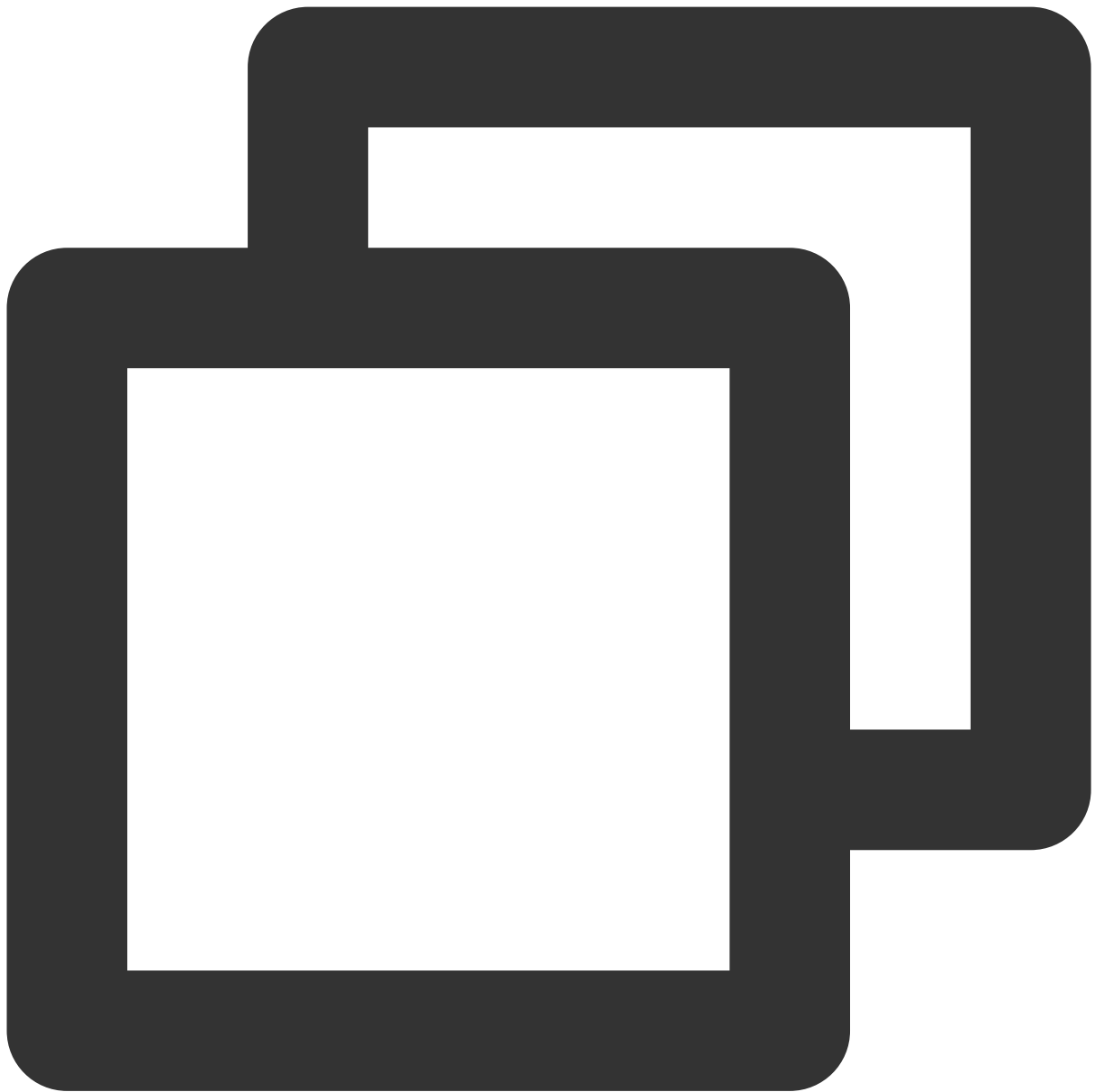
最近更新时间：2024-06-27 11:08:34

小程序 SDK 提供以下 API 用于访问最近使用的小程序列表。



```
/**
 * 获取最近访问小程序列表
 * @param callback
 */
public static void getRecentList(IRecentMiniCallback callback)
```


示例代码：



```
TmfMiniSDK.getRecentList(new IRecentMiniCallback() {  
    @Override  
    public void get(List<MiniApp> data) {  
        //data为最近访问的小程序列表  
    }  
});
```

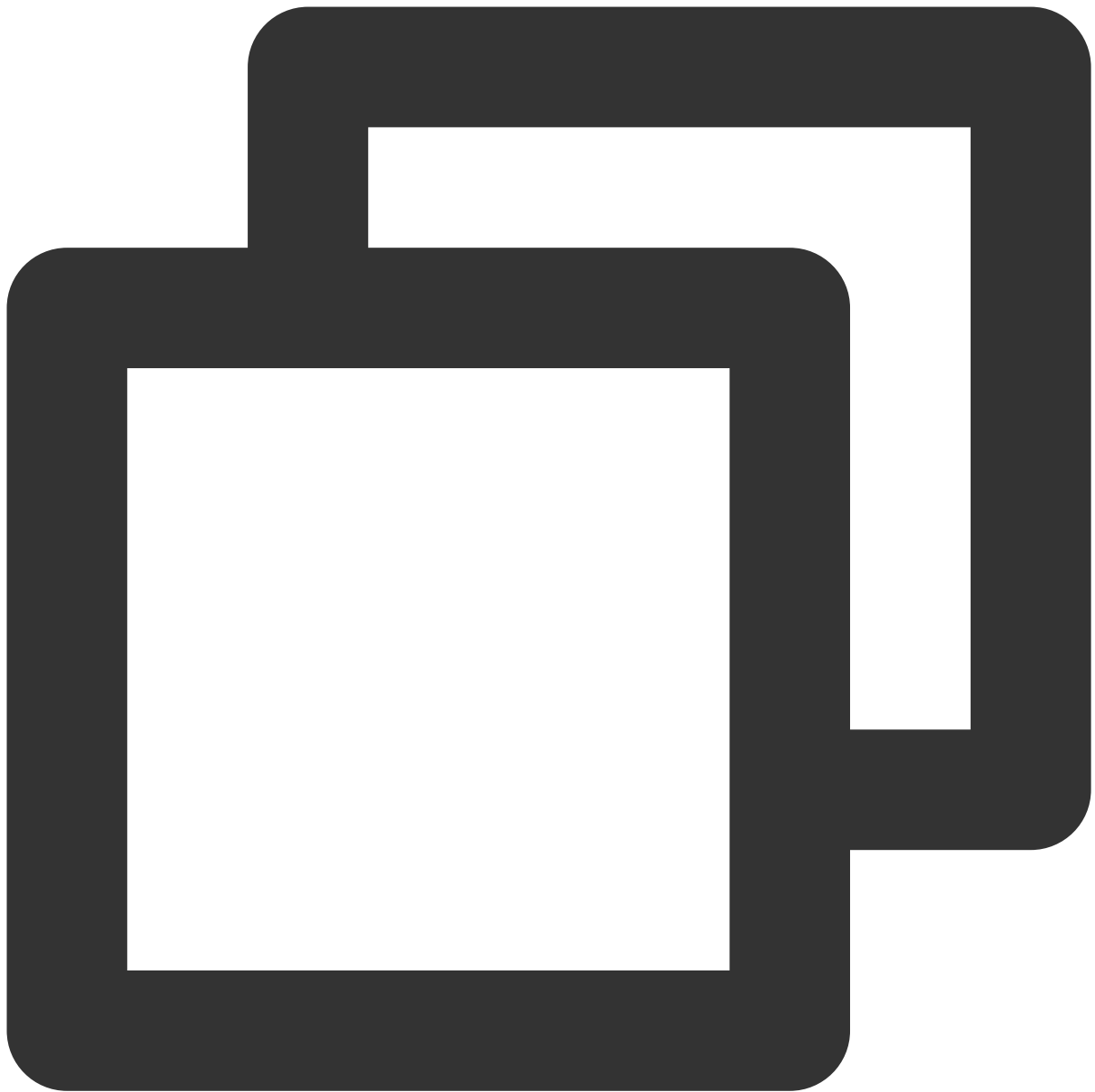

预下载小程序包

最近更新时间：2024-06-27 11:08:07

由于小程序在打开时会同步检测和更新版本，小程序在打开过程中可能存在等待的过程；为了降低小程序打开的等待时间，优化用户体验，提供了如下的接口用于预下载小程序包。

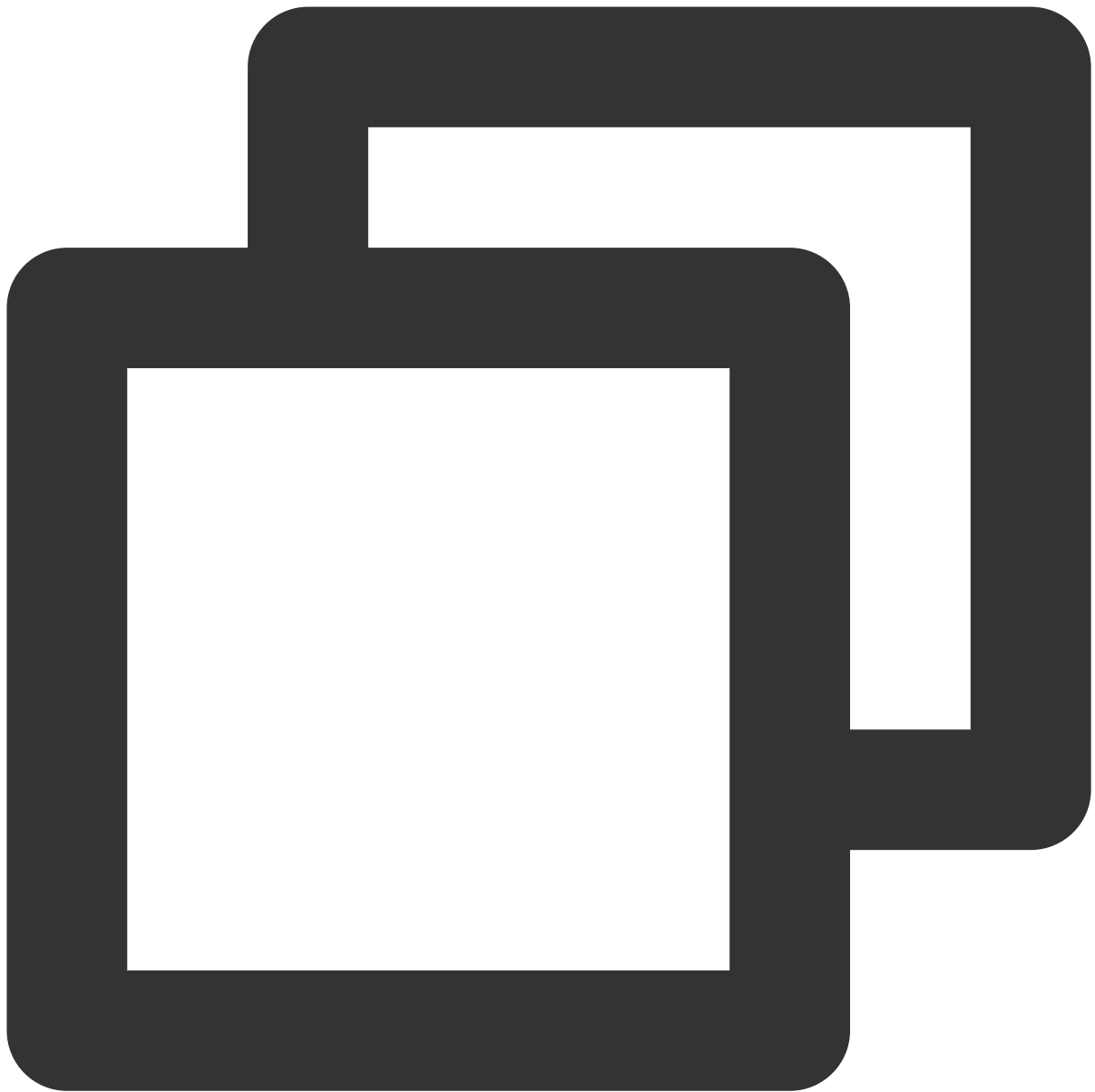
预下载单个小程序主包

API 说明：



```
/**
 * 预下载小程序主包
 *
 * @param preDownloadInfo 预下载信息
 * @param callback 下载回调
 */
public static void preDownloadPkg (PreDownloadInfo preDownloadInfo, IDownloadCallbac
```

示例代码：



```
PreDownloadInfo downloadInfo = new PreDownloadInfo("appId");
TmfMiniSDK.preDownloadPkg(downloadInfo, new IDownloadCallback() {
    @Override
    public void onFinish(DownloadInfo downloadInfo) {

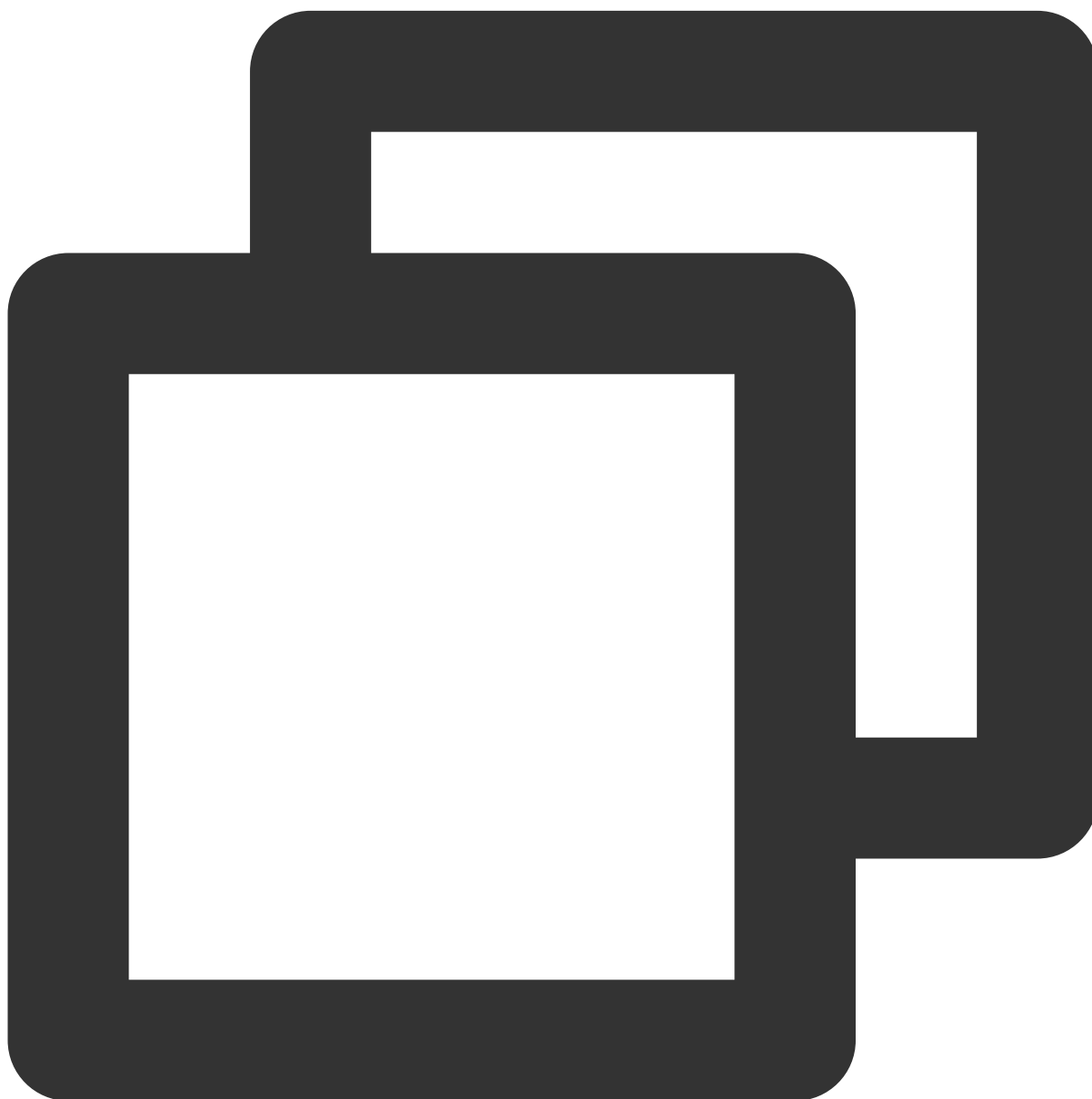
    }

    @Override
    public void onError(DownloadInfo downloadInfo) {
```

```
}  
});
```

预下载多个小程序主包

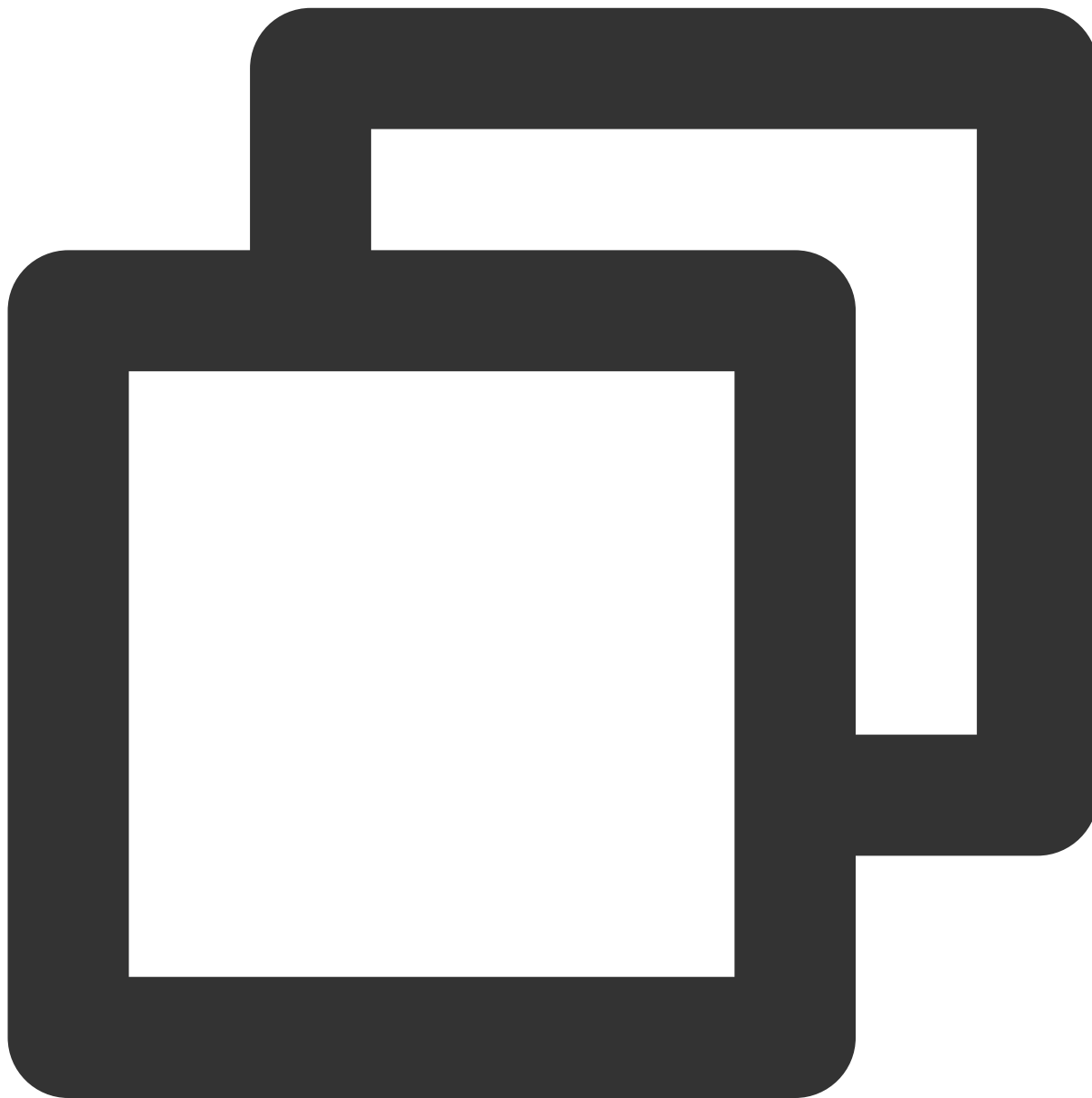
API 说明：



```
/**  
 * 预下载小程序主包
```

```
* @param preDownloadInfos 预下载信息
* @param callback 下载回调
*/
public static void preDownloadPkg(List<PreDownloadInfo> preDownloadInfos, IDownload
```

示例代码：



```
PreDownloadInfo downloadInfo = new PreDownloadInfo("appId");
PreDownloadInfo downloadInfo2 = new PreDownloadInfo("appId2");
ArrayList<PreDownloadInfo>infos = new ArrayList<>();
infos.add(downloadInfo);
infos.add(downloadInfo2);
```

```
TmfMiniSDK.preDownloadPkg(infos, new IDownloadCallback() {  
    @Override  
    public void onFinish(DownloadInfo downloadInfo) {  
  
    }  
  
    @Override  
    public void onError(DownloadInfo downloadInfo) {  
  
    }  
});
```

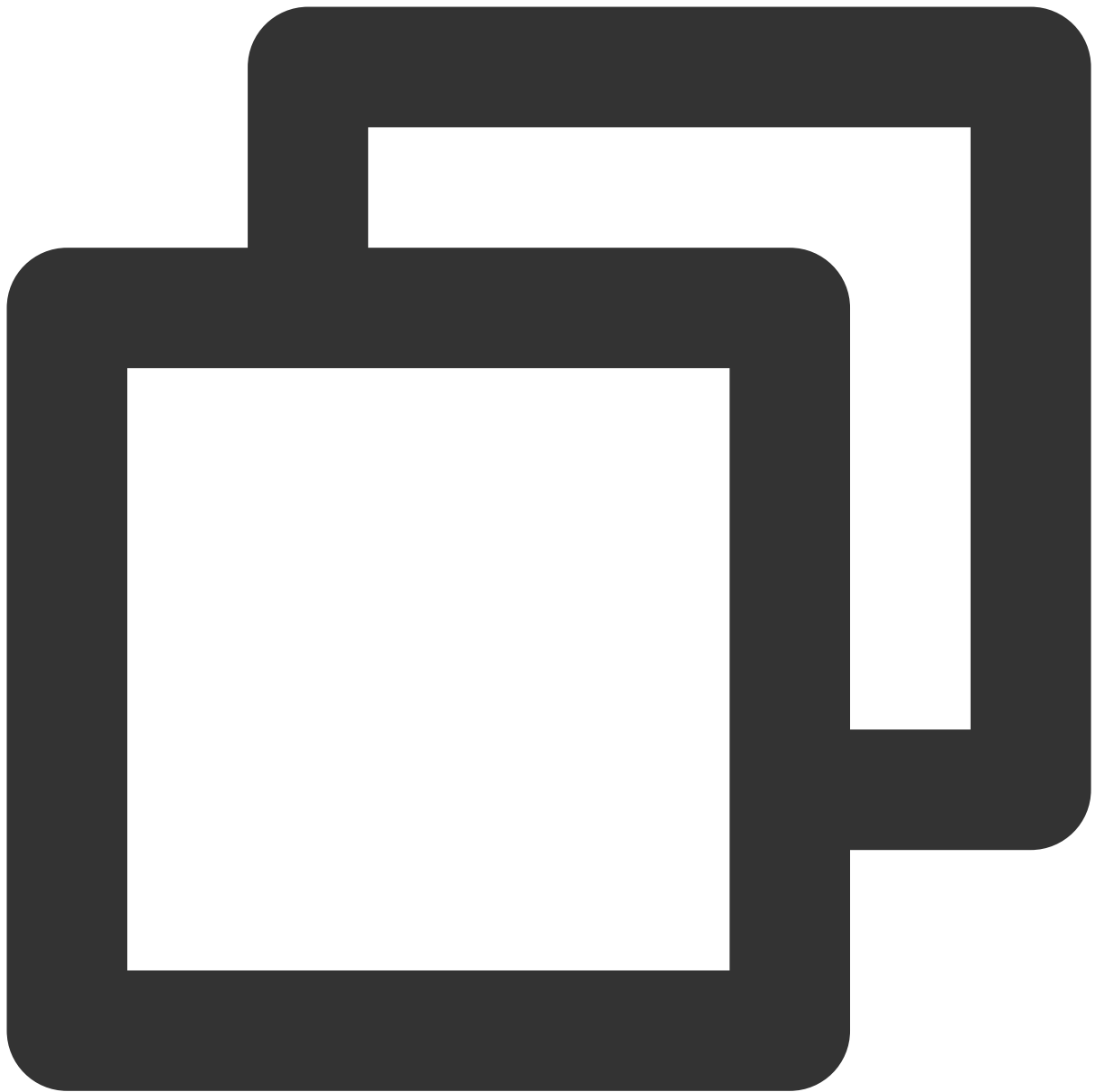

多进程交互

最近更新时间：2024-06-27 11:07:53

小程序 SDK 会为每个打开的小程序创建一个独立的进程，所以宿主应用的进程和小程序的进程是隔离的，小程序 SDK 提供了统一的接口用于小程序进程和宿主进程通信。

判断是否小程序进程

API 说明：



```
/**
 * 当前运行的进程是否是小程序进程
 * @param context
 * @return
 */
public static boolean isMiniProcess(Context context)
```

宿主进程调用小程序进程的插件

1. 实现一个子进程的插件扩展

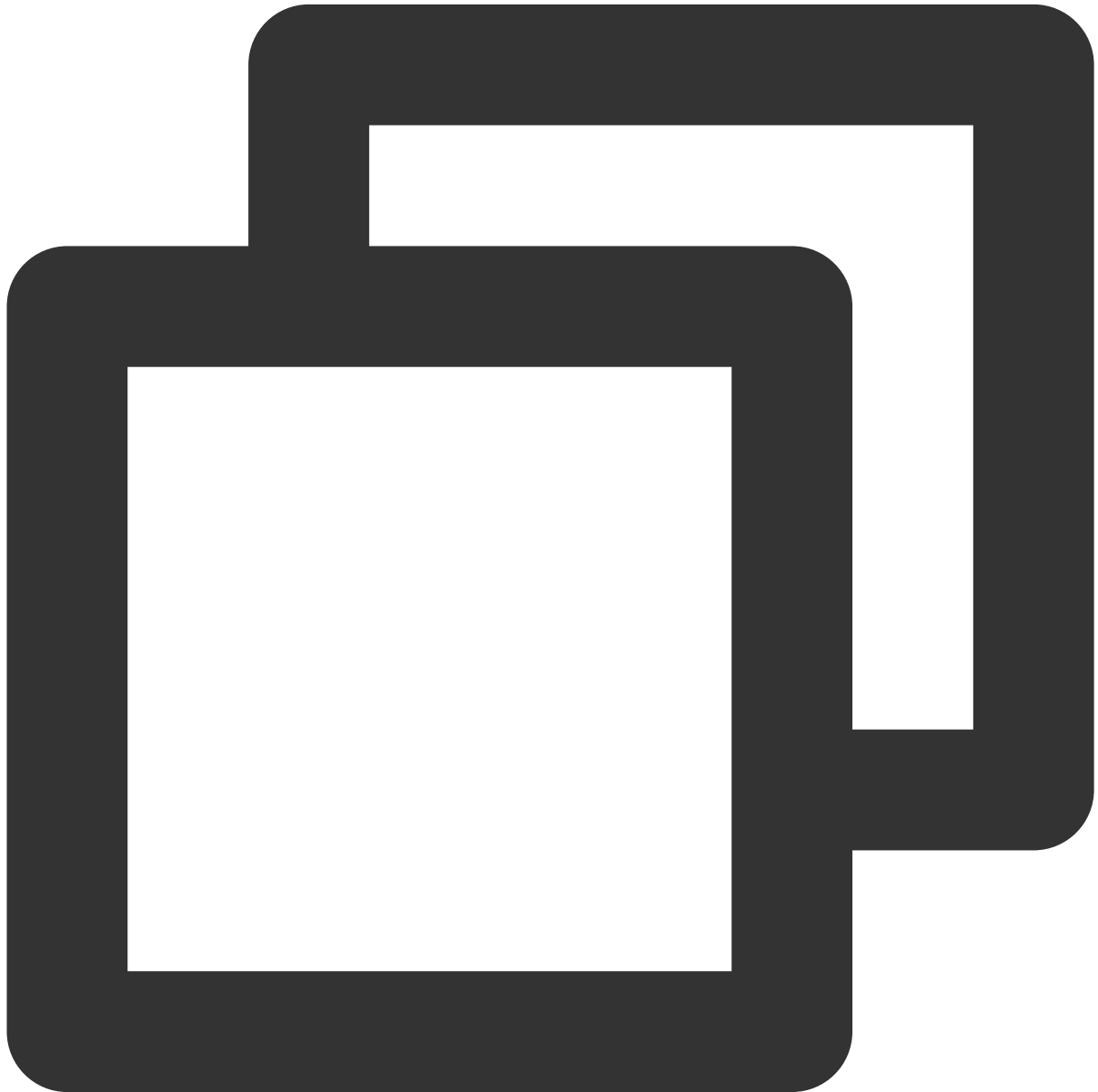
实现步骤说明：

继承 BaseIpcPlugin。

使用注解 @IpcMiniPlugin 修饰定义的类。

使用注解 @IpcEvent 修饰 invoke 方法，并定义插件事件名。

示例代码：



```
@IpcMiniPlugin
public class MiniProcessIpcPlugin extends BaseIpcPlugin {
    public static final String EVENT = "MiniProcessPlugin";
}
```

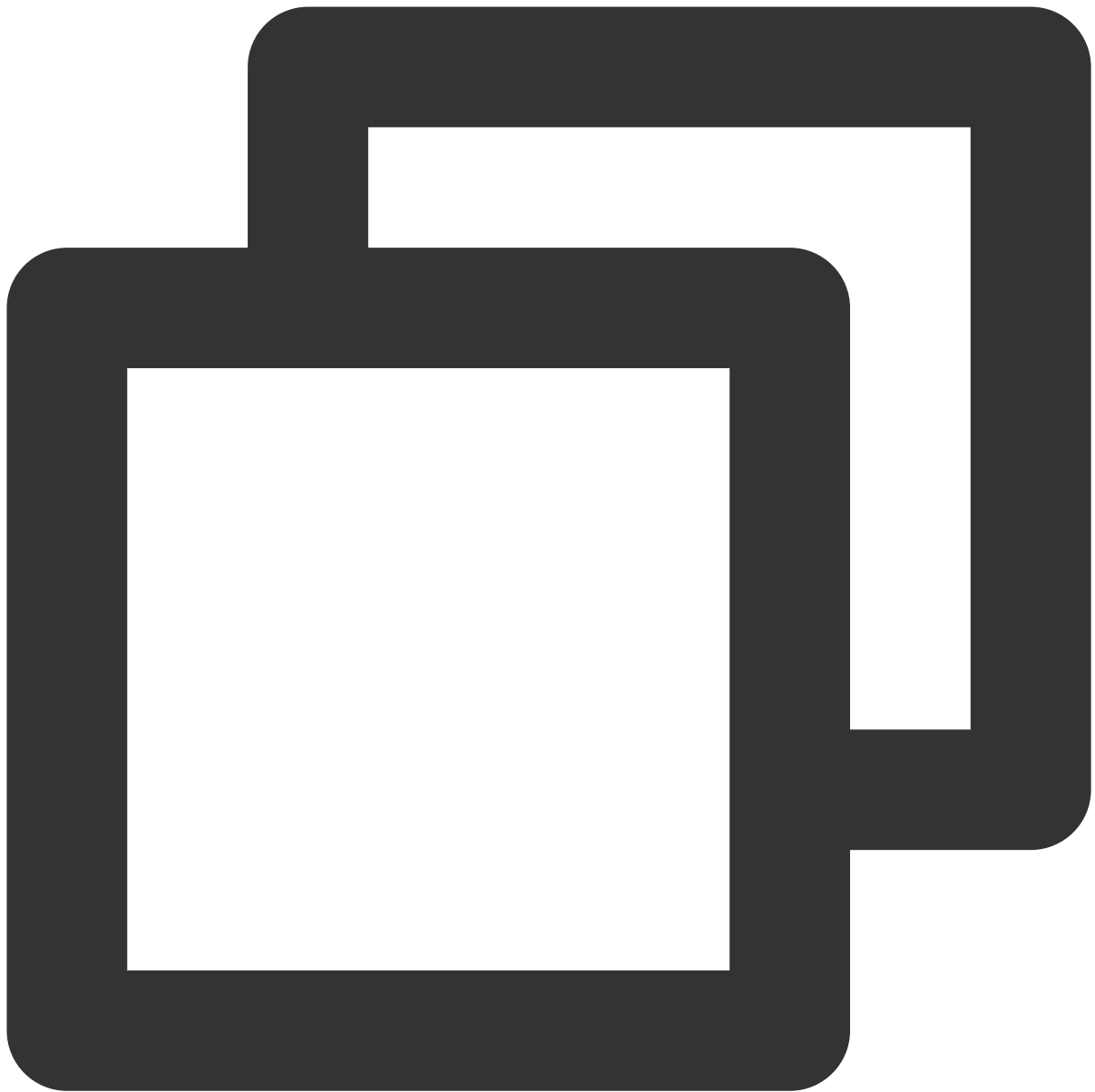
```
@Override
@IpcEvent(EVENT)
public void invoke(IpcRequestEvent req) {

    String value = req.data.getString("key");

    Log.d(ModuleApplet.TAG, "current process:" + ProcessUtil.getProcessName(req
    Bundle resp = new Bundle();
    resp.putString("key", "i am ok");
    req.ok(resp);
}
}
```

2. 宿主进程中调用小程序进程的插件

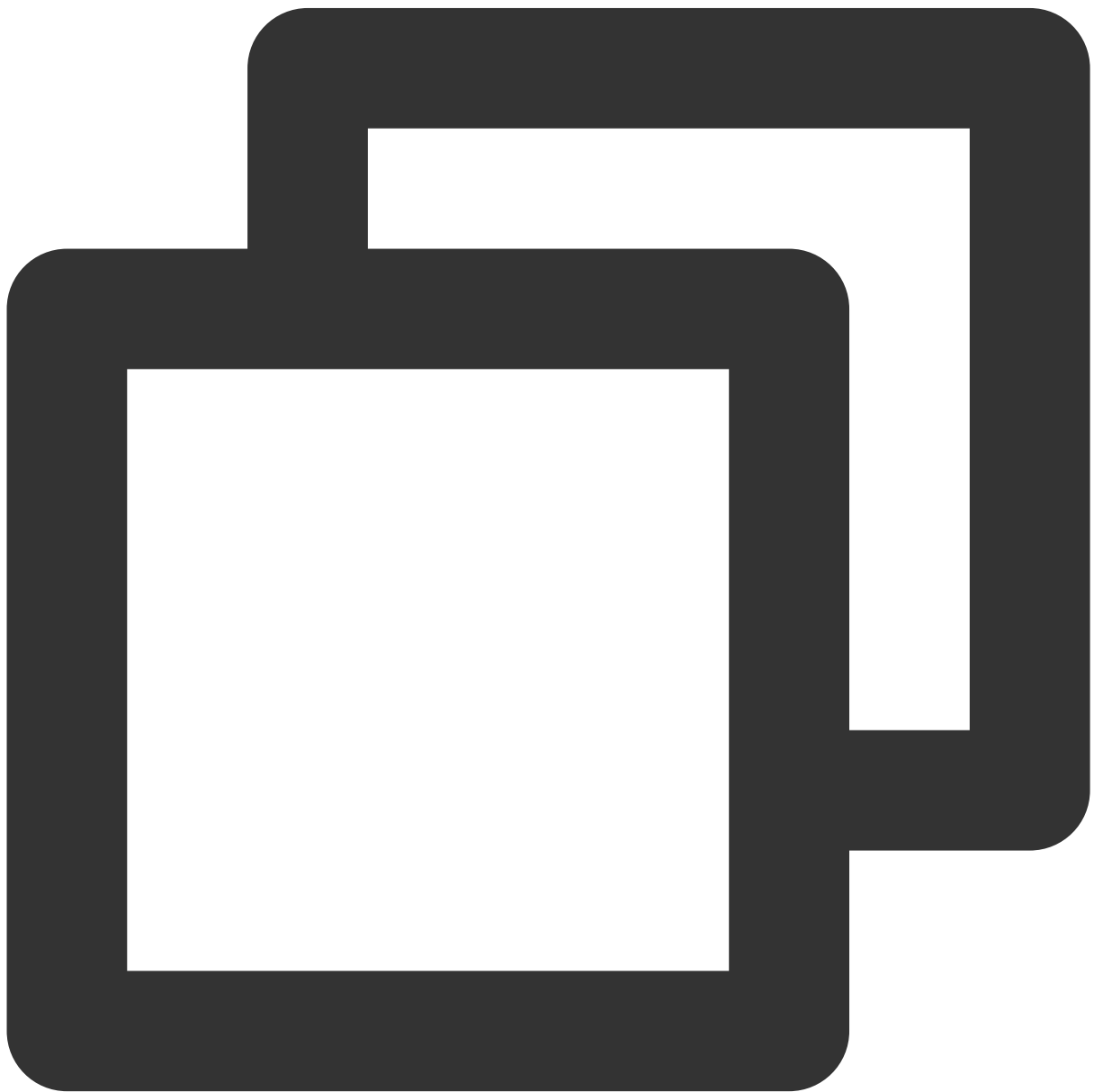
API 说明



```
/**
 * 调用小程序进程Plugin插件（调用正式版小程序），此方法只能在主进程中被调用
 *
 * @param appId    小程序id
 * @param eventId
 * @param request
 * @param callback
 */
public static void callMiniProcessPlugin(String appId, String eventId, Bundle request,
/**
```

```
* 调用小程序进程Plugin插件（调用开发\\预览小程序），此方法只能在主进程中被调用
*
* @param appId      小程序id
* @param appVerType 小程序类型,参考 MiniApp
* @param eventId
* @param request
* @param callback
*/
public static void callMiniProcessPlugin(String appId, int appVerType, String event
```

示例代码：



```
TmfMiniSDK.callMiniProcessPlugin("", MiniProcessIpcPlugin.EVENT, bundle, new IpcCal
    @Override
    public void result(boolean isSuccess, Bundle response) {
        Log.d(ModuleApplet.TAG, "current process:" + ProcessUtil.getProcessName(Mod
    }
});
```

小程序进程调用宿主进程插件

1. 实现一个宿主进程的扩展插件

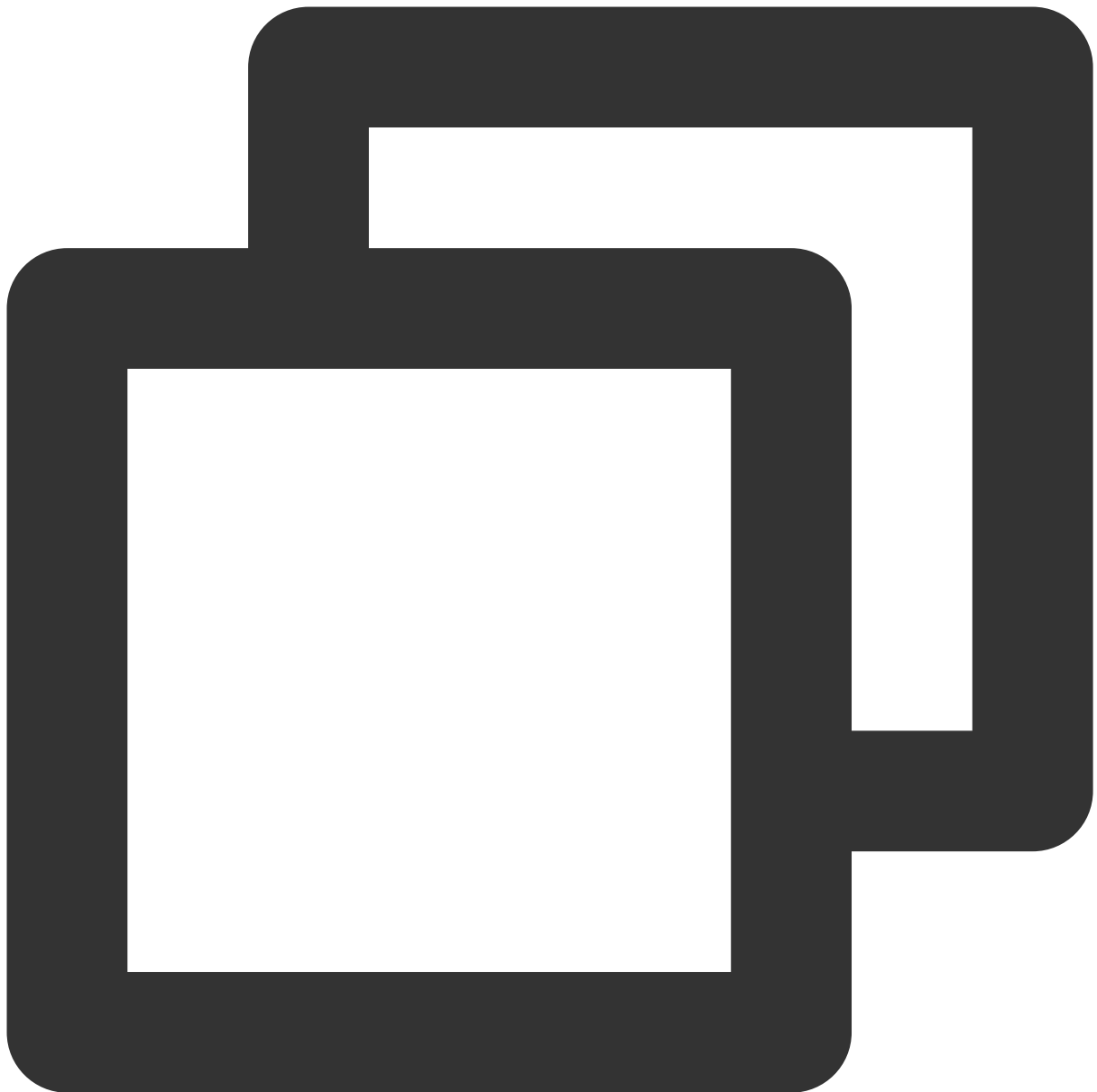
实现步骤说明：

继承 `BaseIpcPlugin`。

使用注解 `@IpcMainPlugin` 修饰定义的类。

使用注解 `@IpcEvent` 修饰 `invoke` 方法，并定义插件事件名。

示例代码：



```
@IpcMainPlugin
public class MainProcessIpcPlugin extends BaseIpcPlugin {
    public static final String EVENT = "MainProcessPlugin";

    @Override
    @IpcEvent(EVENT)
    public void invoke(IpcRequestEvent req) {
        String value = req.data.getString("key");

        Log.d(ModuleApplet.TAG, "current process|" + ProcessUtil.getProcessName(req)
        Bundle resp = new Bundle();
```



```
        resp.putString("key", "i am ok");  
        req.ok(resp);  
    }  
}
```

2. 小程序进程中调用宿主进程插件

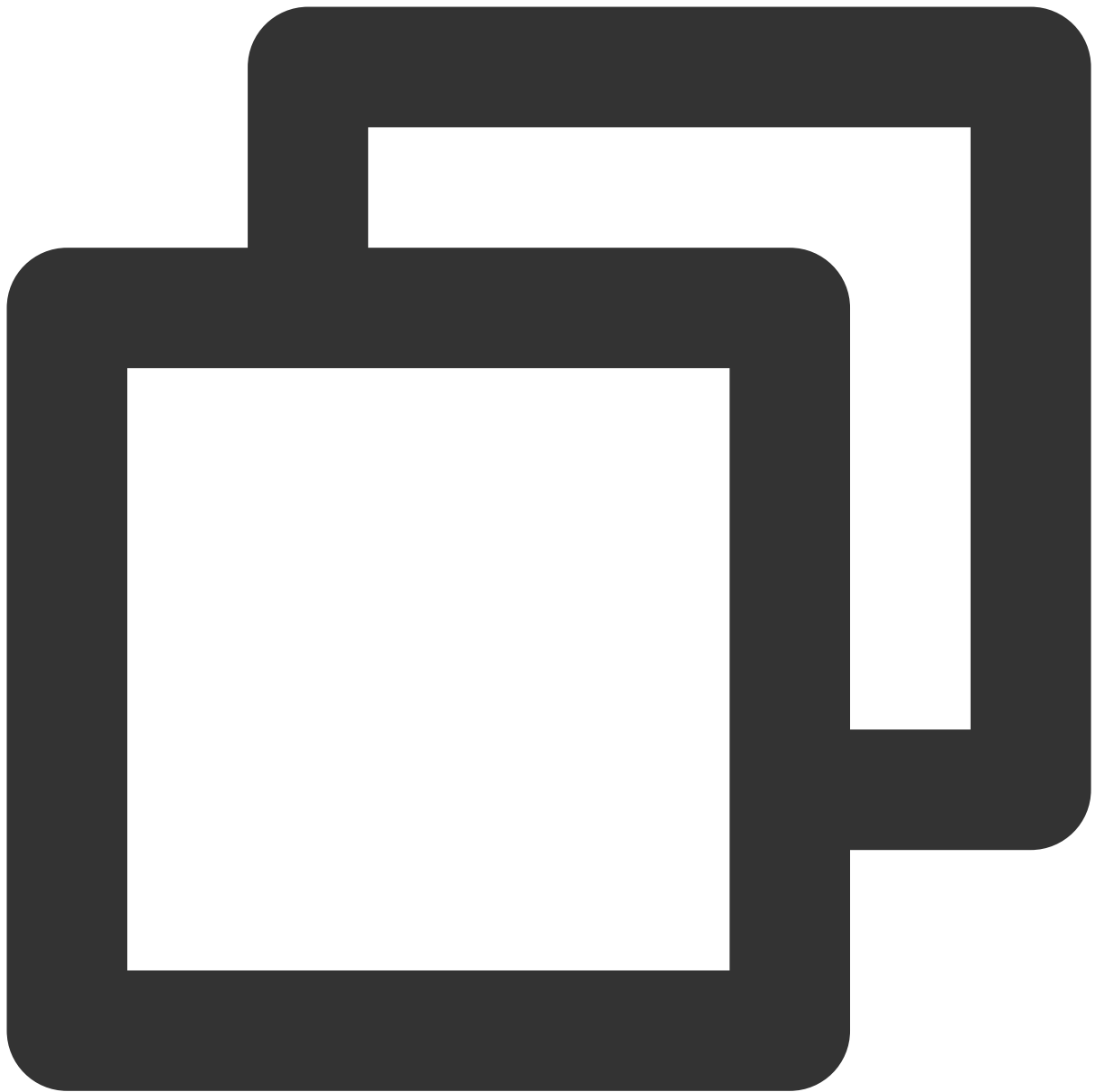
API 说明

参数说明：

参数 eventId，表示宿主进程定义的事件名称。

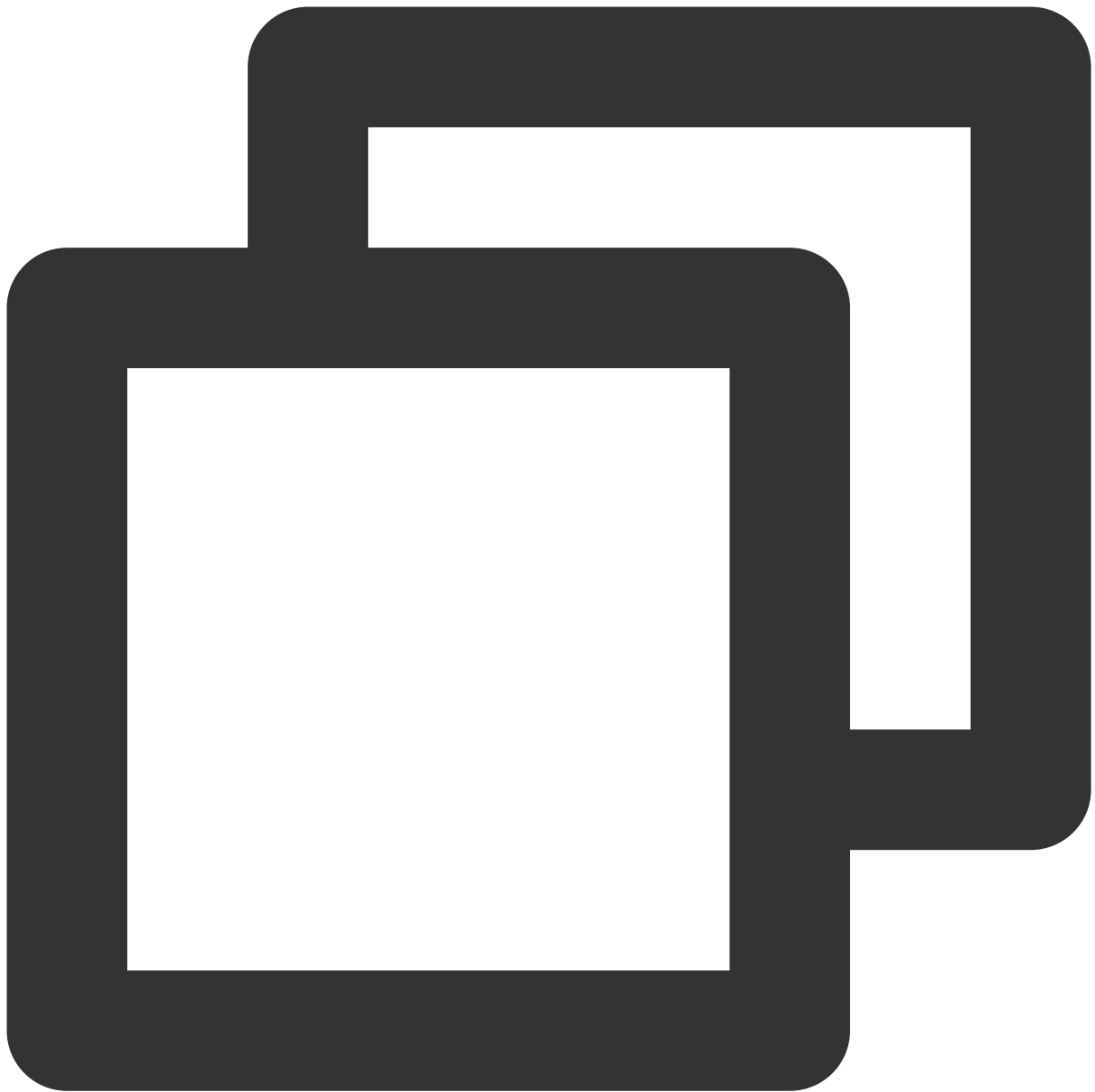
参数 request，表示小程序进程传递给宿主进程插件的参数。

参数ipcCallBack，用于接收宿主进程回调。



```
/**
 * 调用主进程Plugin插件，此方法只能在小程序进程中被调用
 *
 * @param eventId
 * @param request
 * @param callback
 */
public static void callMainProcessPlugin(String eventId, Bundle request, IpcCallbac
```

示例代码：



```
TmfMiniSDK.callMainProcessPlugin(MainProcessIpcPlugin.EVENT, bundle, new IpcCallbac
@Override
public void result(boolean isSuccess, Bundle response) {
    Log.d(ModuleApplet.TAG, "current process:" + ProcessUtil.getProcessName(Mod
}
});
```

小程序文件管理

最近更新时间：2024-06-27 11:00:45

通过小程序文件管理 API，可以实现小程序的文件路径和宿主应用本地文件路径的转换及路径创建。

说明：

小程序文件路径：以 `wxfile://` 开头的小程序开发者使用路径，是一个路径标识符号，每一个以

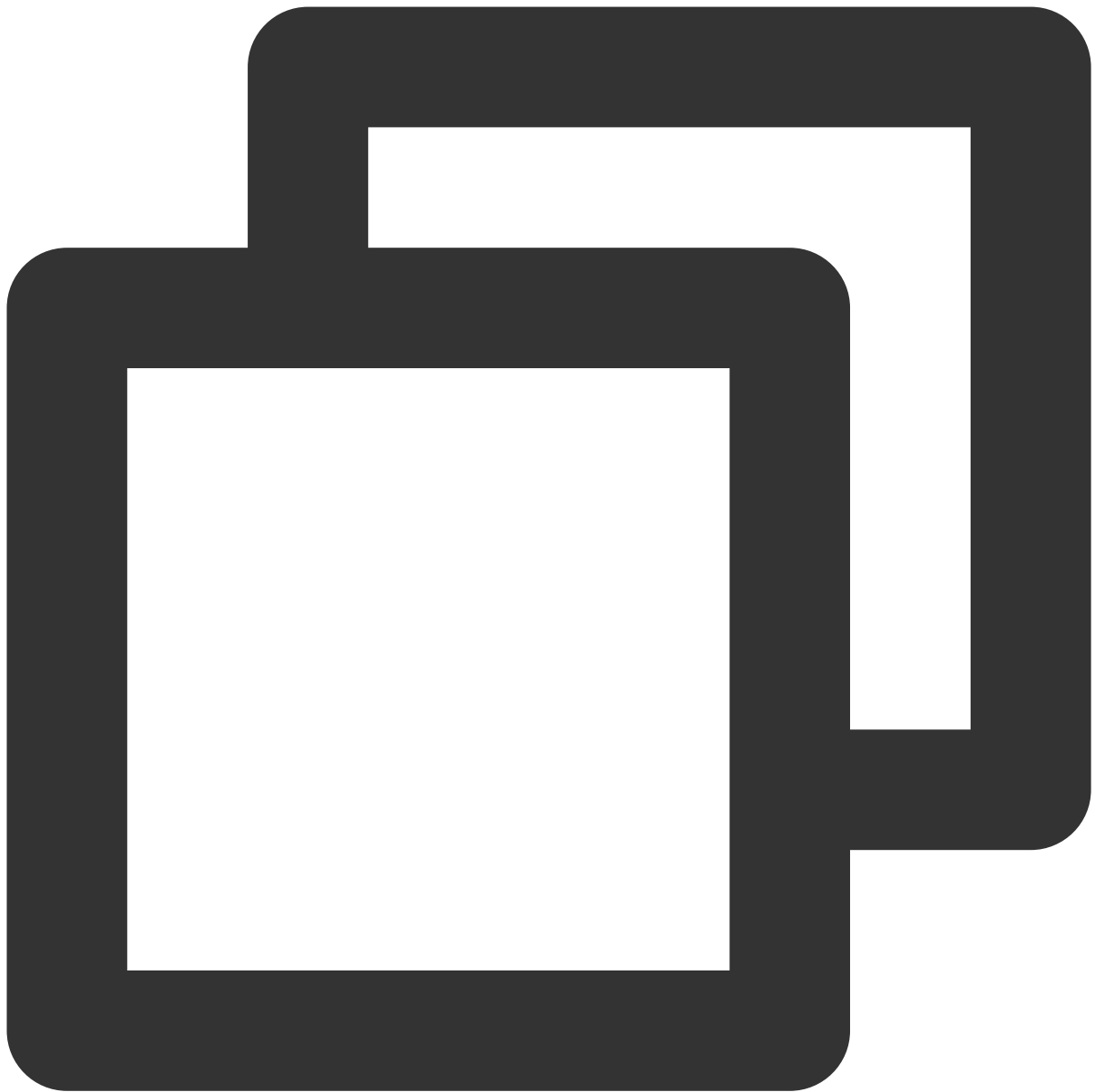
`wxfile://` 开头的路径都会被小程序SDK映射到一个宿主应用内的本地文件路径。

宿主应用本地文件路径：宿主应用和设备存储中的绝对路径，

如：`/data/data/com.tencent.miniapp.demo/app_T1701421723ASSNID/2121/files/mini/`。

获取 `IMiniAppFileManager`

小程序文件统一由 `IMiniAppFileManager` 管理，需通过小程序上下文 `IMiniAppContext` 获取，`IMiniAppContext` 可以从 `BaseJsPlugin` 获取。

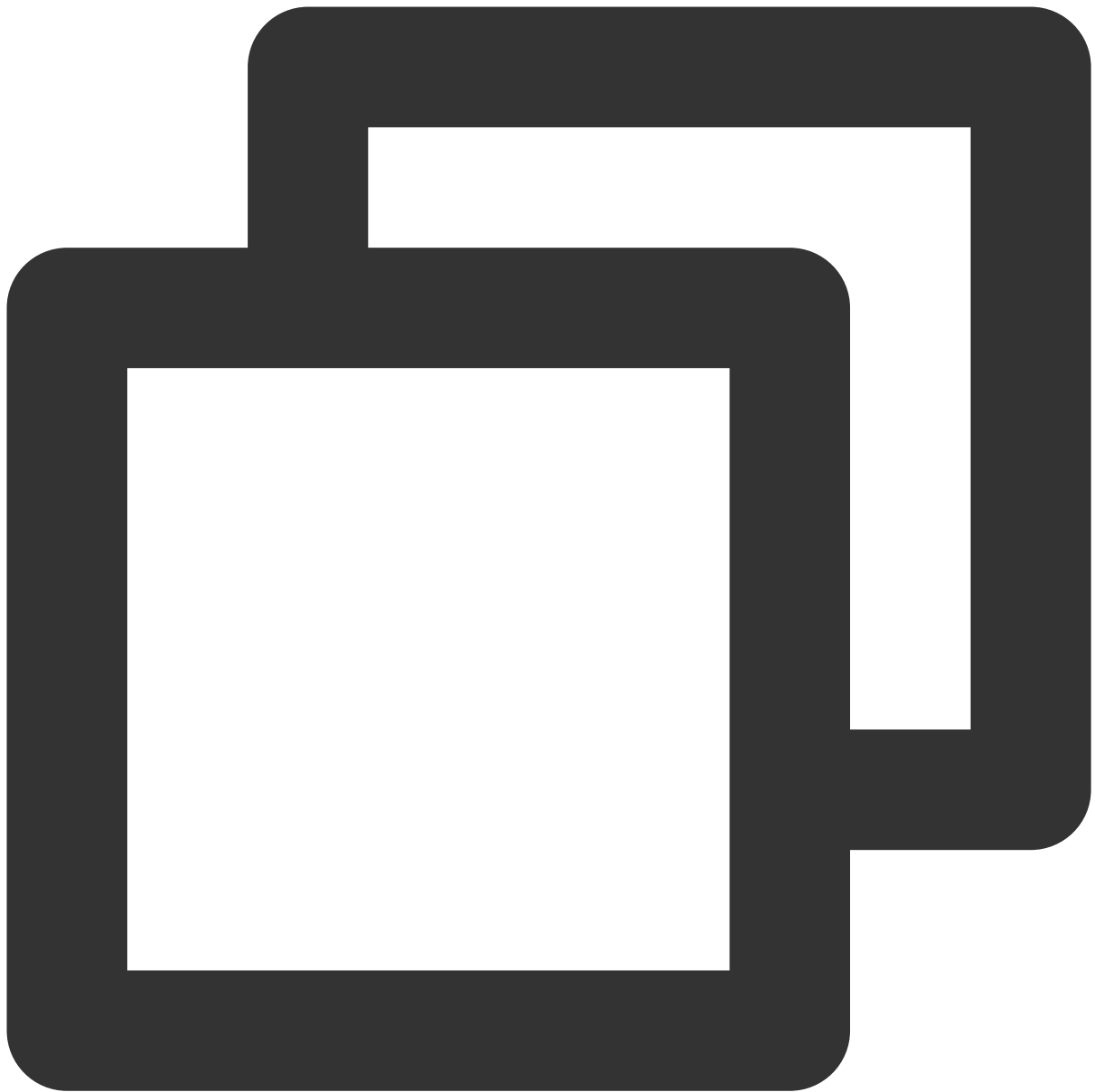


```
IMiniAppFileManager fileManager = mMiniAppContext.getManager(IMiniAppFileManager.cl
```

在小程序临时目录创建文件

SDK 支持原生在小程序的缓存目录创建文件，然后返回文件的本地路径，以供宿主使用。

示例代码：

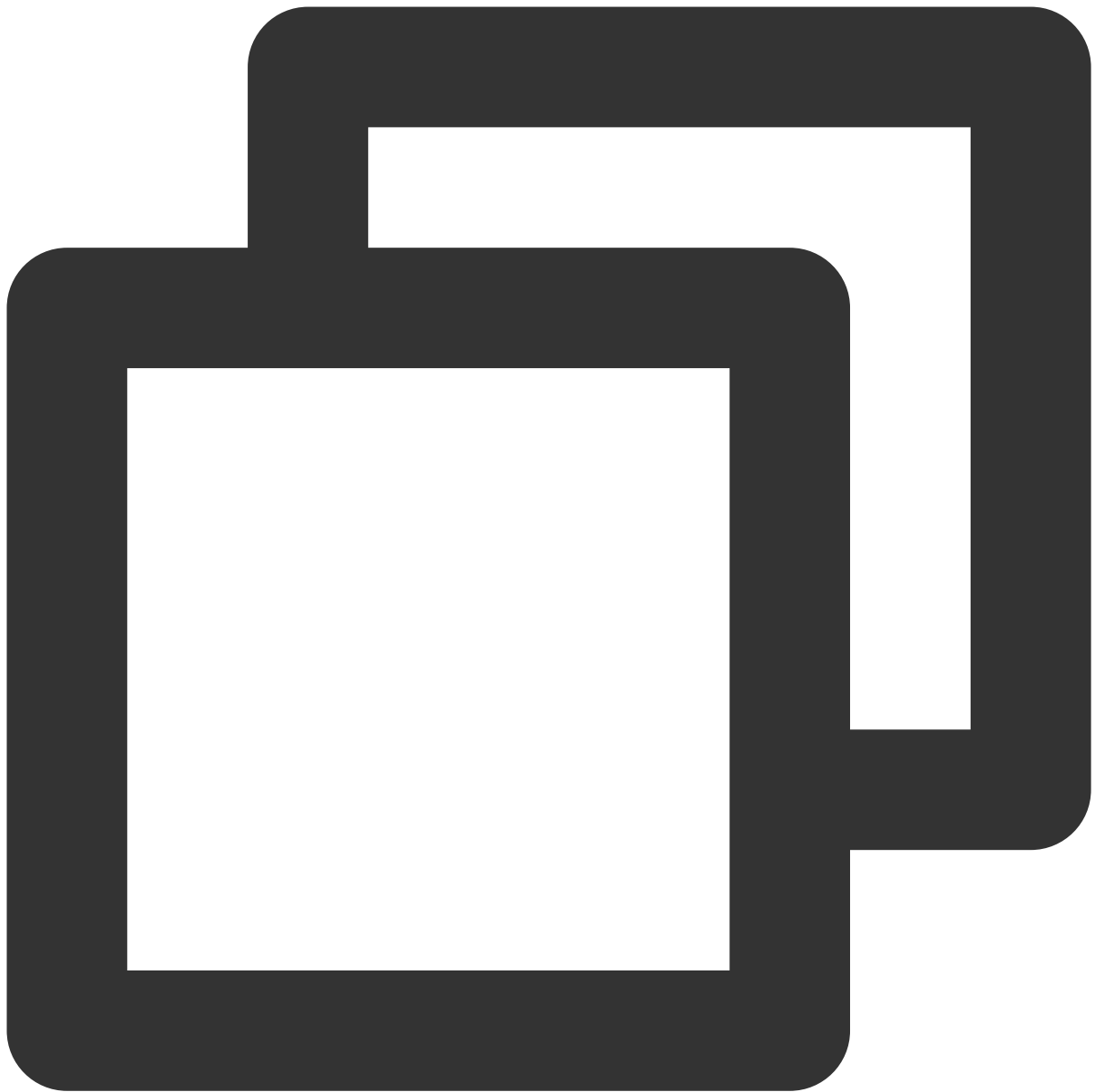


```
String tmpPath = fileManager.getTmpPath(".jpg");
```

绝对路径转换成 wxfile 路径

SDK 支持原生在小程序的缓存目录中创建的文件路径，转换成小程序使用的文件路径返回，供小程序内部使用。

示例代码：

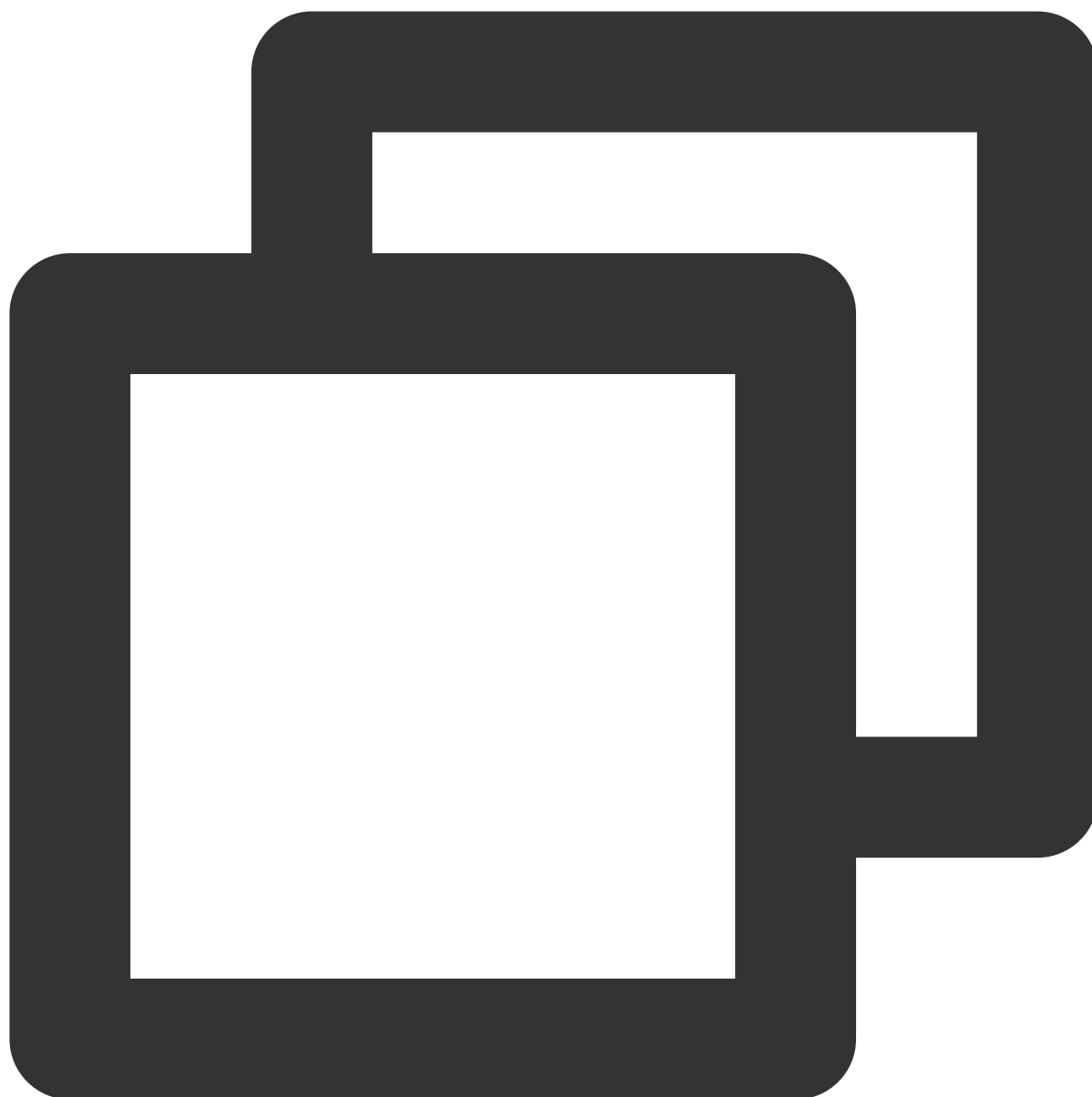


```
String wxfile = fileManager.getWxFilePath(path);
```

wxfile 路径转换为绝对路径

SDK 支持将换成目录中的 `wxfile` 路径转换为本地完整路径，供宿主使用。

示例代码：



```
String path = fileManager.getAbsolutePath(wxfile);
```


自定义小程序能力

自定义小程序 API

最近更新时间：2024-06-27 11:00:52

如果在小程序中需要调用某些宿主应用提供的能力，在小程序 SDK 未实现或无法实现时，开发者可以通过注册扩展（自定义）API 来实现相关能力，使得小程序能够调用宿主应用中提供的自定义 API 能力。

宿主应用实现自定义 API

宿主应用通过继承 `BaseJsPlugin` 实现自定义的小程序API能力。

说明：

继承 `BaseJsPlugin` 并用注解进行定义 `@JsPlugin(secondary = true)` ；

定义一个方法，方法只能有一个参数且参数必须是 `RequestEvent` 类型；

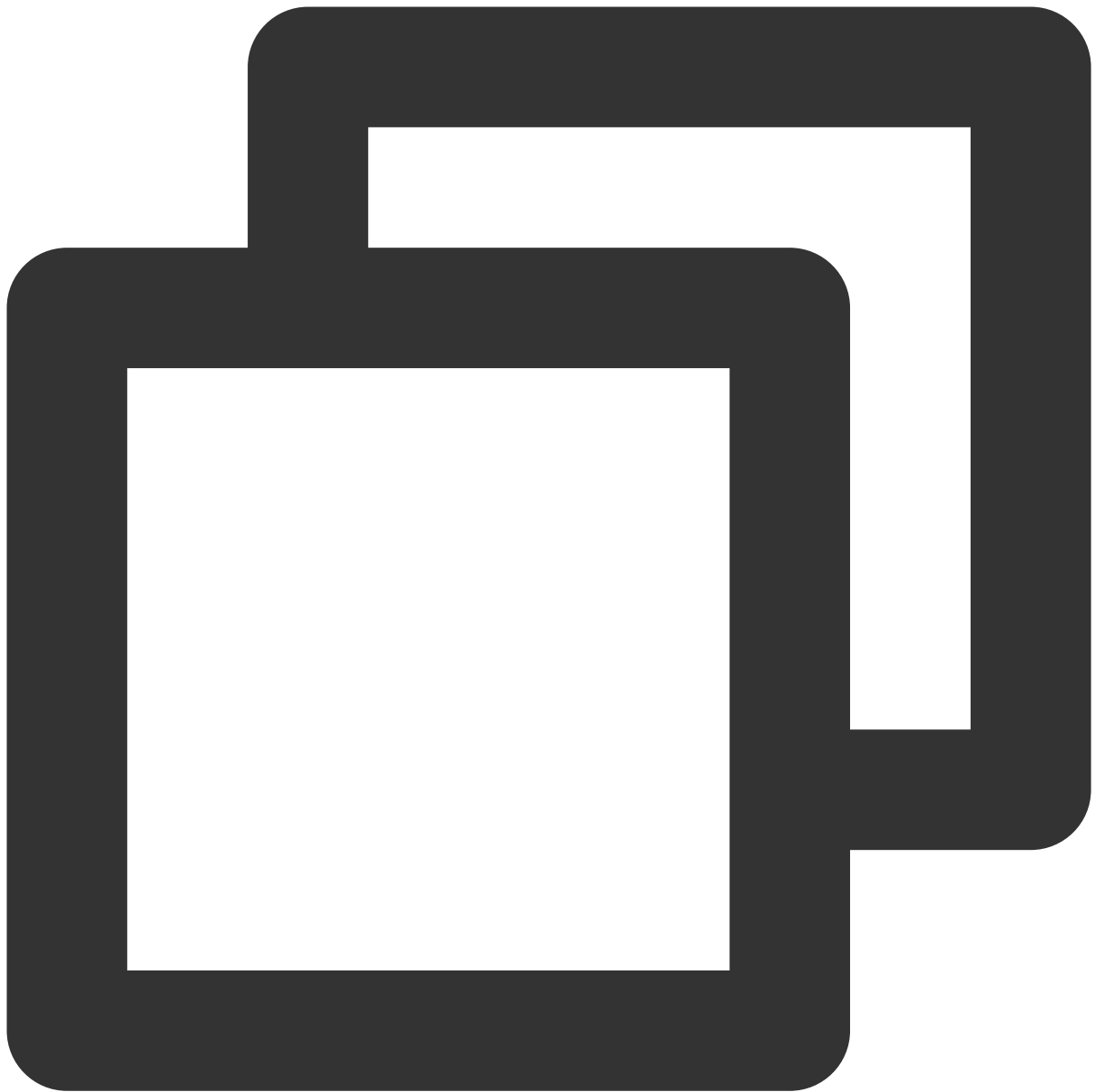
然后在方法上定义注解 `@JsEvent("事件名")`，当小程序 js 调用“事件名”时就会调用到 `@JsEvent` 修饰的对应方法；

`@JsEvent` 支持定义多个事件名；

支持同步或异步返回数据（同一事件只能选择一种方式）；

可以通过调用 `sendState` 给小程序端多次返回中间状态，`sendState` 调用结束后必须调用 `ok` 或 `fail` 标识整个流程结束。

示例代码：



```
@JsPlugin(secondary = true)
public class CustomPlugin extends BaseJsPlugin {
    @JsEvent("customAsyncEvent")
    public void custom(final RequestEvent req) {
        //获取参数
        //req.jsonParams
        //异步返回数据
        //req.fail();
        //req.ok();
        JSONObject jsonObject = new JSONObject();
        try {
```

```
        jsonObject.put("key", "test");
    } catch (JSONException e) {
        e.printStackTrace();
    }
    req.ok(jsonObject);
}

@jsEvent("customSyncEvent")
public String custom1(final RequestEvent req) {
    //获取参数
    //req.jsonParams
    //同步返回数据
    JSONObject jsonObject = new JSONObject();
    try {
        jsonObject.put("key", "value");
    } catch (JSONException e) {
        throw new RuntimeException(e);
    }
    return req.okSync(jsonObject);
}

/**
 * 测试覆盖系统API
 * @param req
 */
@jsEvent("getAppBaseInfo")
public void getAppBaseInfo(final RequestEvent req) {
    //获取参数
    //req.jsonParams
    //异步返回数据
    //req.fail();
    //req.ok();
    JSONObject jsonObject = new JSONObject();
    try {
        jsonObject.put("key", "test");
    } catch (JSONException e) {
        e.printStackTrace();
    }
    req.ok(jsonObject);
}

@jsEvent("testState")
public void testState(final RequestEvent req) {

    try {
        //回调中间状态
        req.sendState(req, new JSONObject().put("progress", 1));
    }
}
```

```
        req.sendState(req, new JSONObject().put("progress", 30));
        req.sendState(req, new JSONObject().put("progress", 60));
        req.sendState(req, new JSONObject().put("progress", 100));
    } catch (JSONException e) {
        e.printStackTrace();
    }

    JSONObject jsonObject = new JSONObject();
    try {
        jsonObject.put("key", "test");
        req.ok(jsonObject);
    } catch (JSONException e) {
        throw new RuntimeException(e);
    }
}
```

配置自定义 API

参考如下的模版配置自定义 API 的参数信息：

说明：

配置模版中，自定义 API 的配置信息会被放在如下的 json 文件中。

json 文件外层的 extApi 字段用于存放自定义 API 数组，extApi 中的每一个对象表示一个自定义 API 的配置信息；

json 文件外层的 overwriteWxApi 字段用于配置自定义事件是否覆盖默认的小程序 API 实现。

设置为 true 是表示覆盖 SDK 内置的小程序 API。

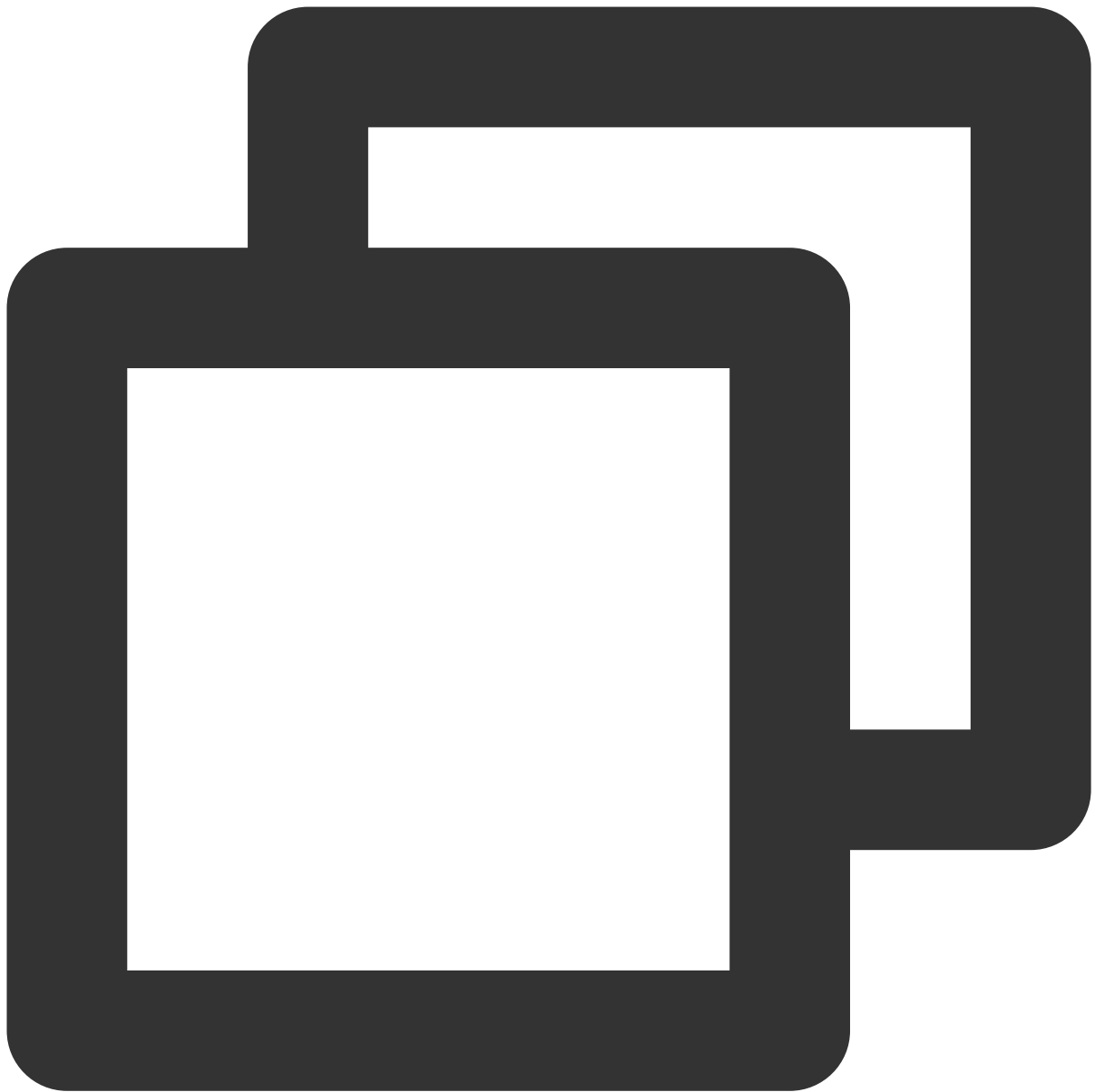
每个自定义的 API 配置需要包含有如下三个关键字：

name：自定义 API 的事件名称，需要与 @JsEvent 定义的事件名称一致；

sync：是否同步调用，需要与“创建API”示例数据返回方式保持一致；

parmas：自定义 API 的参数描述。

模版如下：

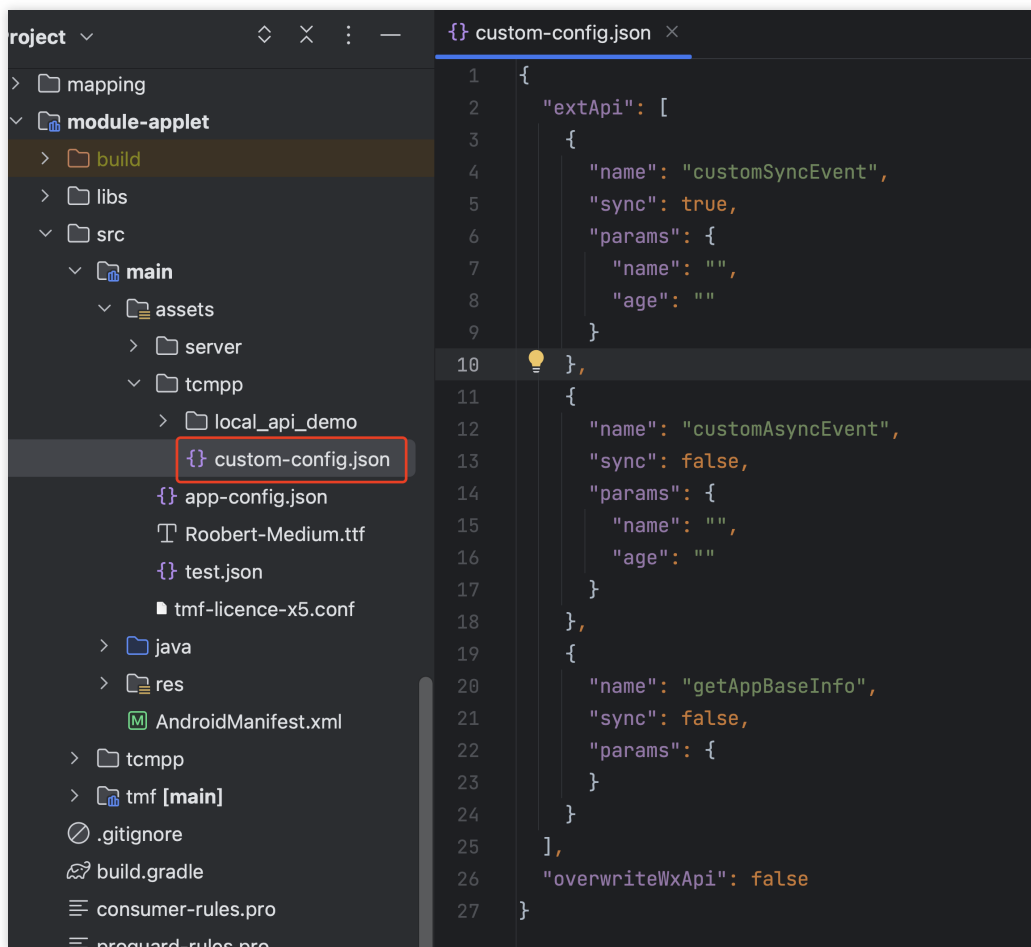


```
{
  "extApi": [
    {
      //事件名，与“创建API”示例中@jsEvent定义的事件保持一致
      "name": "customSyncEvent",
      //是否同步调用，与“创建API”示例数据返回方式保持一致
      "sync": true,
      //定义参数
      //a.json格式可以嵌套
      //b.字符串参数value设置为""即可
      //c.数字参数value设置为0即可
    }
  ]
}
```

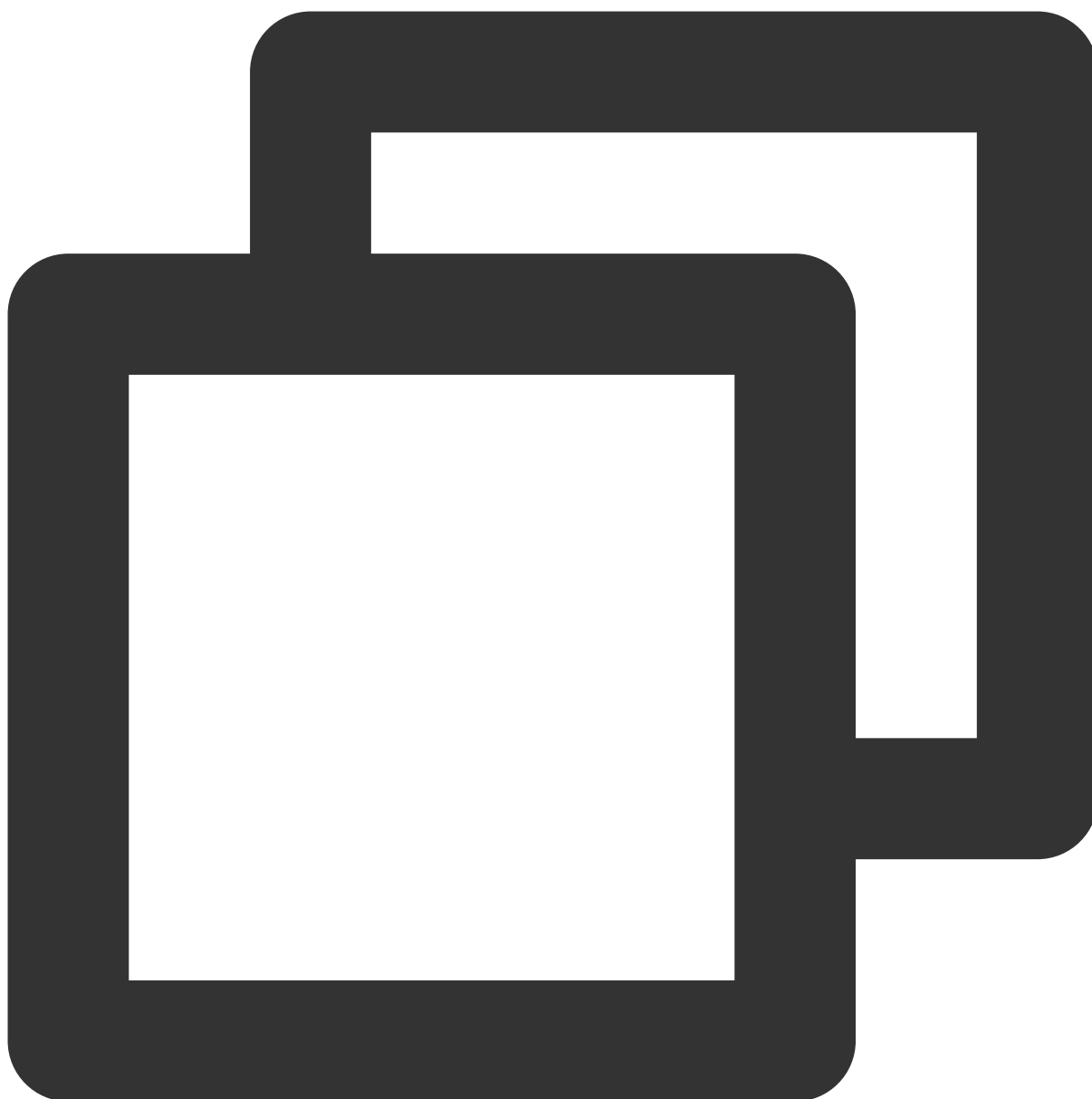
```
"params": {
  "name": "",
  "age": 0,
  "object": {
    "key": "",
  }
},
{
  "name": "customAsyncEvent2",
  "sync": false,
  "params": {
    "name": "",
    "age": ""
  }
},
//true：当自定义的API事件名与小程序SDK内置的方法名，会覆盖SDK内置的
//API，最终调用会调用的开发者自定义的API中
"overwriteWxApi": false
}
```

指定自定义 API 的配置文件路径

将定义的 API 配置文件存放到工程的 **assets** 目录下（文件名和路径开发者可以自己定义）。



代码中指定自定义 API 配置文件的路径信息



```
@ProxyService(proxy = MiniAppProxy.class)
public class MiniAppProxyImpl extends BaseMiniAppProxyImpl {
    @Override
    public MiniConfigData configData(Context context, int configType, JSONObject pa
        if(configType == MiniConfigData.TYPE_CUSTOM_JSAPI) {
            //自定义JsApi配置
            MiniConfigData.CustomJsApiConfig customJsApiConfig = new MiniConfigData
            customJsApiConfig.jsApiConfigPath = "tcmpp/custom-config.json";

            return new MiniConfigData
                .Builder()
```



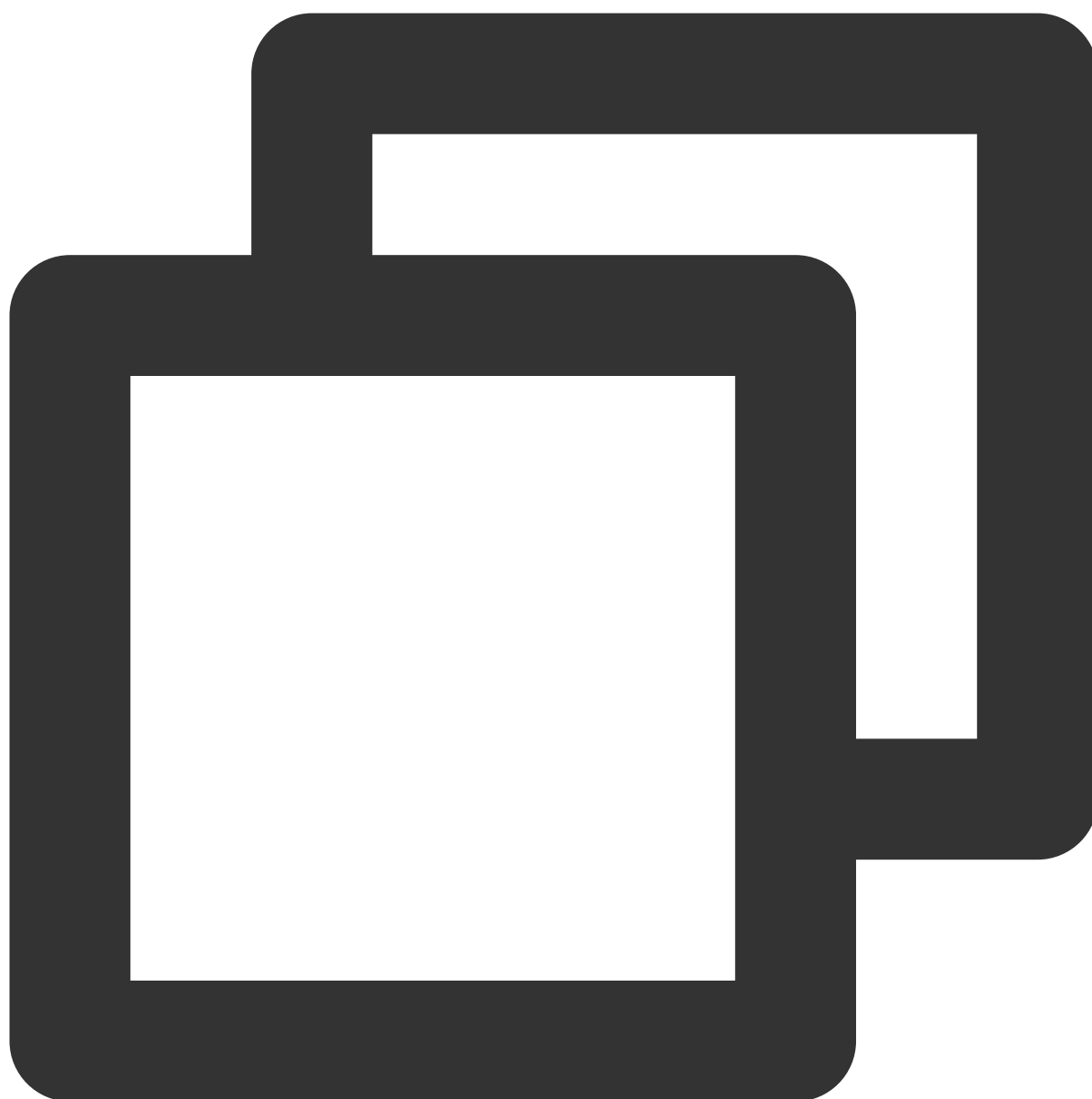
```
        .customJsApiConfig(customJsApiConfig)
        .build();
    }

    return null;
}
```

小程序开发者调用自定义 API

举例：

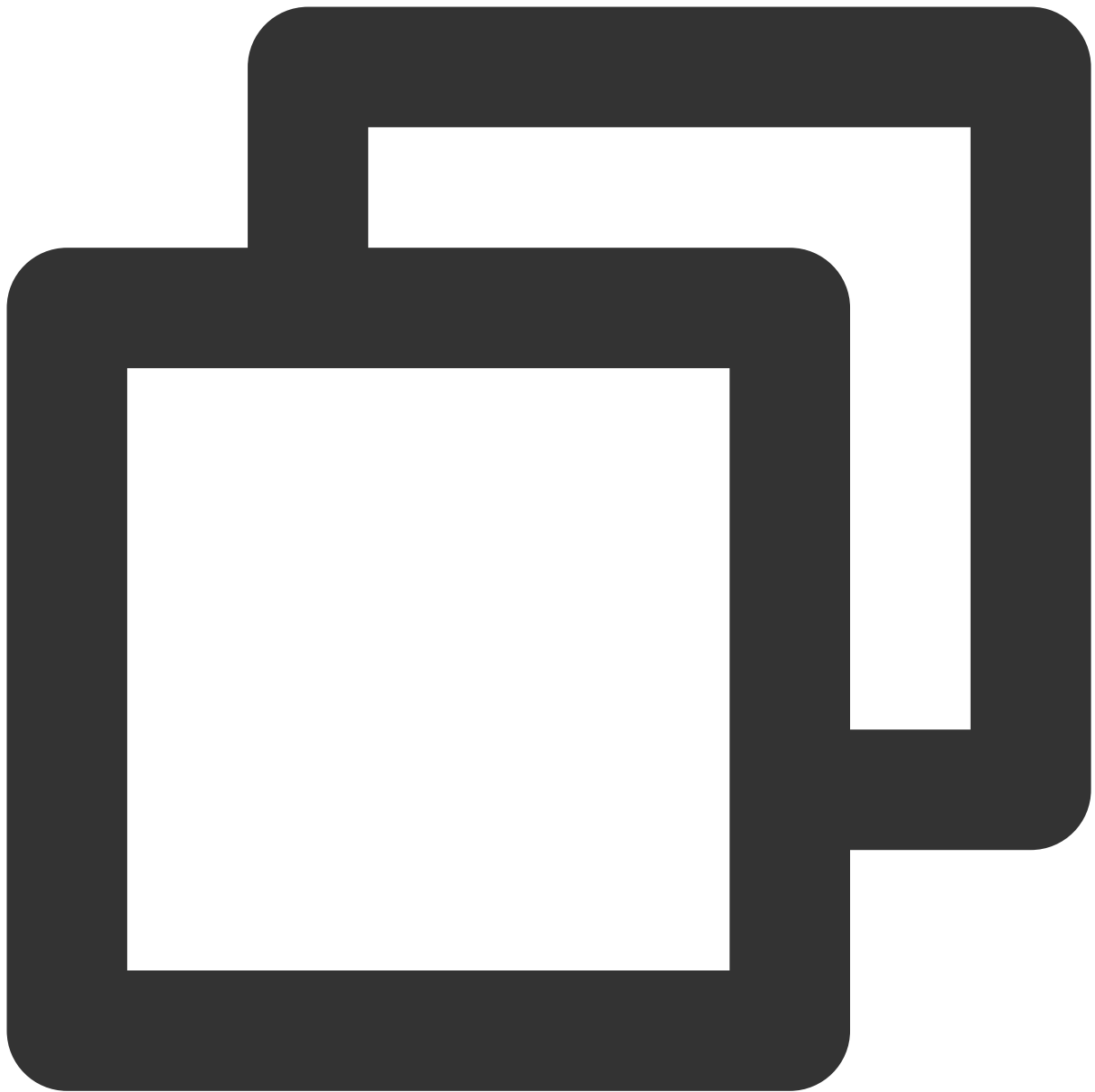
宿主应用自定义 `customAsyncEvent` 和 `getAppBaseInfo` 两个自定义 API，其配置文件如下：



```
{
  "extApi": [
    {
      "name": "customSyncEvent",
      "sync": true,
      "params": {
        "name": "",
        "age": ""
      }
    },
    {
```

```
    "name": "customAsyncEvent",
    "sync": false,
    "params": {
      "name": "",
      "age": ""
    }
  },
  {
    "name": "getAppBaseInfo",
    "sync": false,
    "params": {
    }
  }
  "overwriteWxApi": true
}
```

小程序开发者可以通过如下的方式访问自定义 **API**。



```
wx.customAsyncEvent({ "name": "123", "age": "18" })  
wx.getAppBaseInfo() // 会覆盖系统API，然后调用到自定义API中
```

注意：

由于自定义 API 配置文件中 `overwriteWxApi` 属性的值配置为 `true`，当小程序调用 `wx.getAppBaseInfo` 时，会调用宿主应用自定义 `getAppBaseInfo` 实现。

自定义 API 问题排查

如果自定义 API 出现调用错误，可以参考 [Android 常见问题](#) 进行处理。

自定义小程序 API 示例

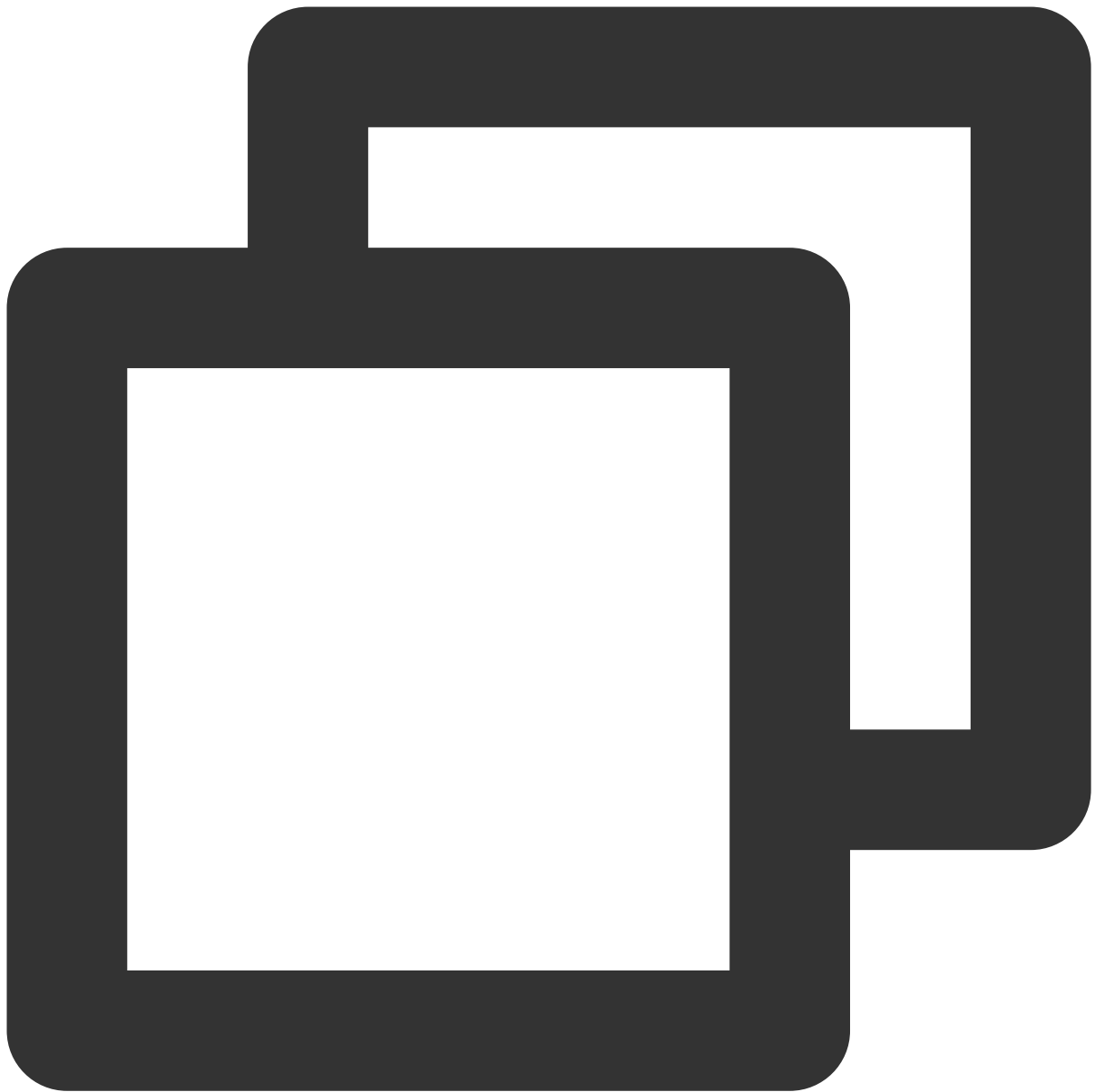
最近更新时间：2024-06-27 10:55:45

在自定义 API 中启动 Activity

插件中启动新 Activity 并获取获取 Activity 返回的数据。

示例代码：

继承 BaseJsPlugin，确定自定义 API 的事件名称：

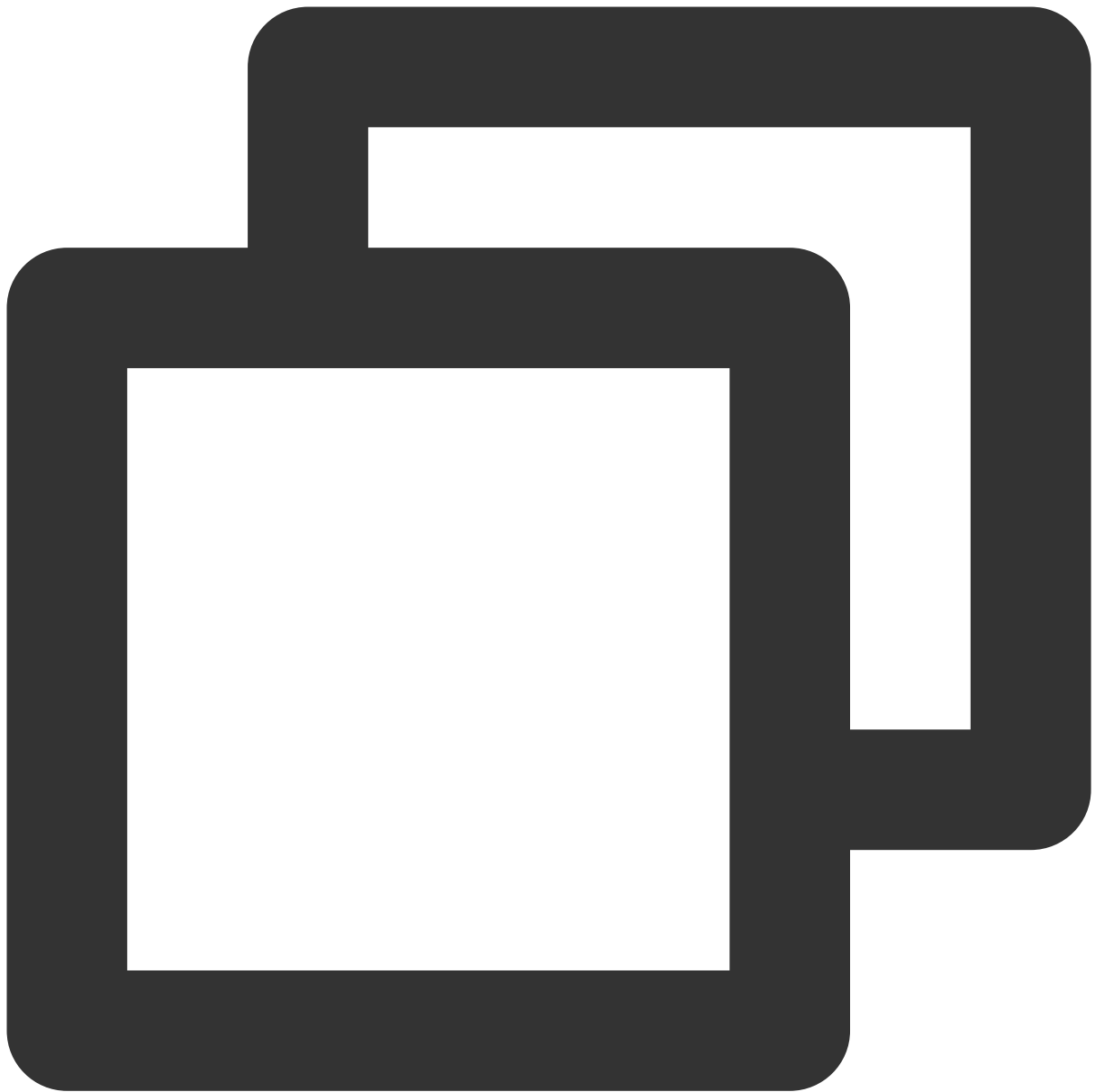


```
@JsPlugin(secondary = true)
public class CustomPlugin extends BaseJsPlugin {
    @JsEvent("testStartActivityForResult")
    public void testStartActivityForResult(final RequestEvent req) {
    }
}
```

添加 Activity 返回监听，并在监听处理完成后移除监听：

注意：

注册监听和移除监听建议配对使用，否则会有内存泄漏的风险。



```
@JsPlugin(secondary = true)
public class CustomPlugin extends BaseJsPlugin {
    @JsEvent("testStartActivityForResult")
    public void testStartActivityForResult(final RequestEvent req) {
        Activity activity = req.activityRef.get();
        TmfMiniSDK.addActivityResultListener(new IActivityResultListener() {
            @Override
            public boolean doOnActivityResult(int requestCode, int resultCode, Intent
                TmfMiniSDK.removeActivityResultListener(this);

            Log.i(ModuleApplet.TAG, data.getStringExtra("key"));
        });
    }
}
```

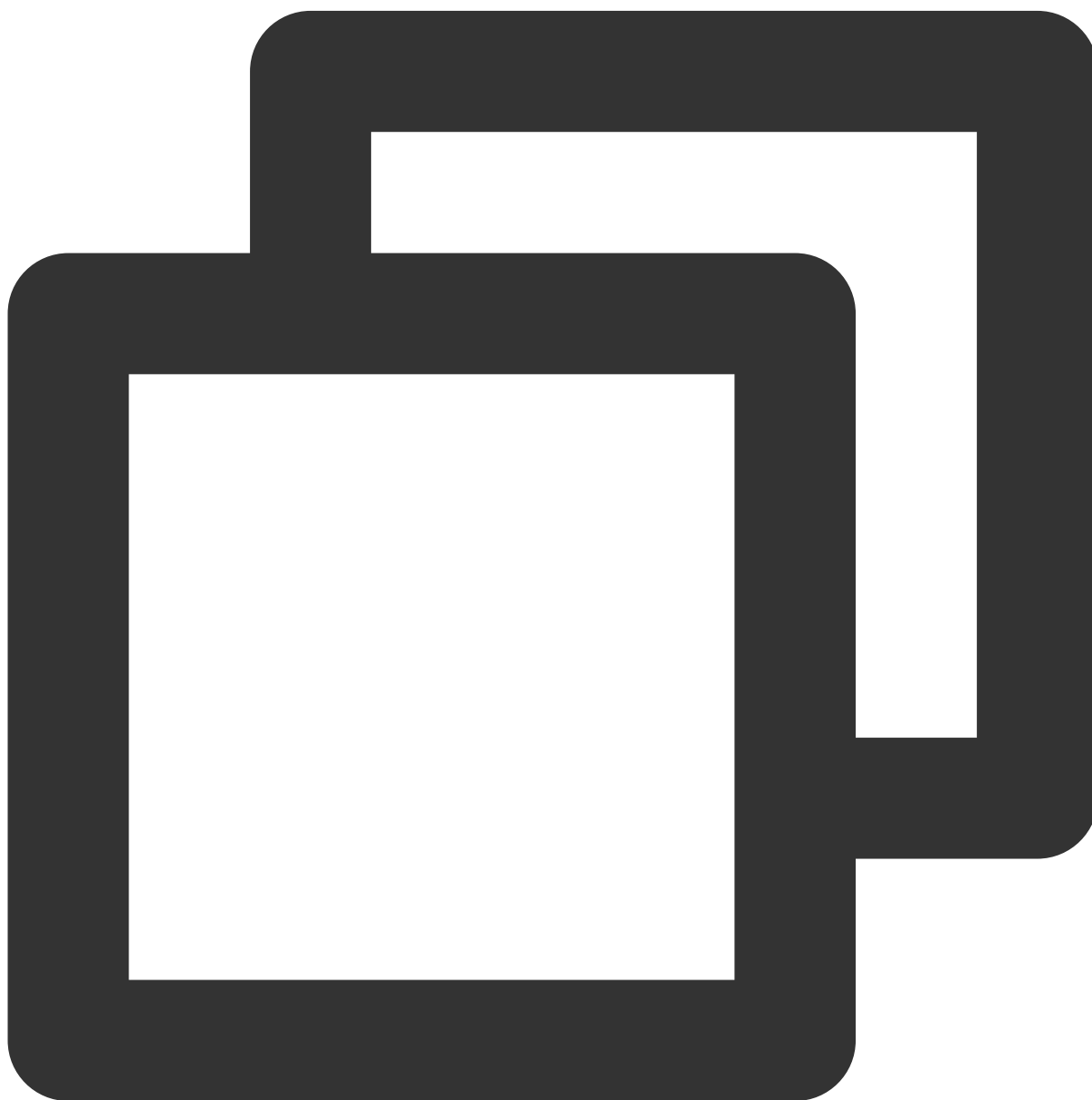


```
        req.ok();  
        return true;  
    }  
    });  
}  
}
```

启动新的 Activity：

注意：

启动 Activity 时，requestCode 必须 ≥ 1000000 ，否则可能与内部 requestCode 冲突，引起未知问题。



```
@JsPlugin(secondary = true)
public class CustomPlugin extends BaseJsPlugin {
    @JsEvent("testStartActivityForResult")
    public void testStartActivityForResult(final RequestEvent req) {
        Activity activity = req.activityRef.get();
        TmfMiniSDK.addActivityResultListener(new IActivityResultListener() {
            @Override
            public boolean doOnActivityResult(int requestCode, int resultCode, Intent data) {
                TmfMiniSDK.removeActivityResultListener(this);

                Log.i(ModuleApplet.TAG, data.getStringExtra("key"));
                req.ok();
                return true;
            }
        });

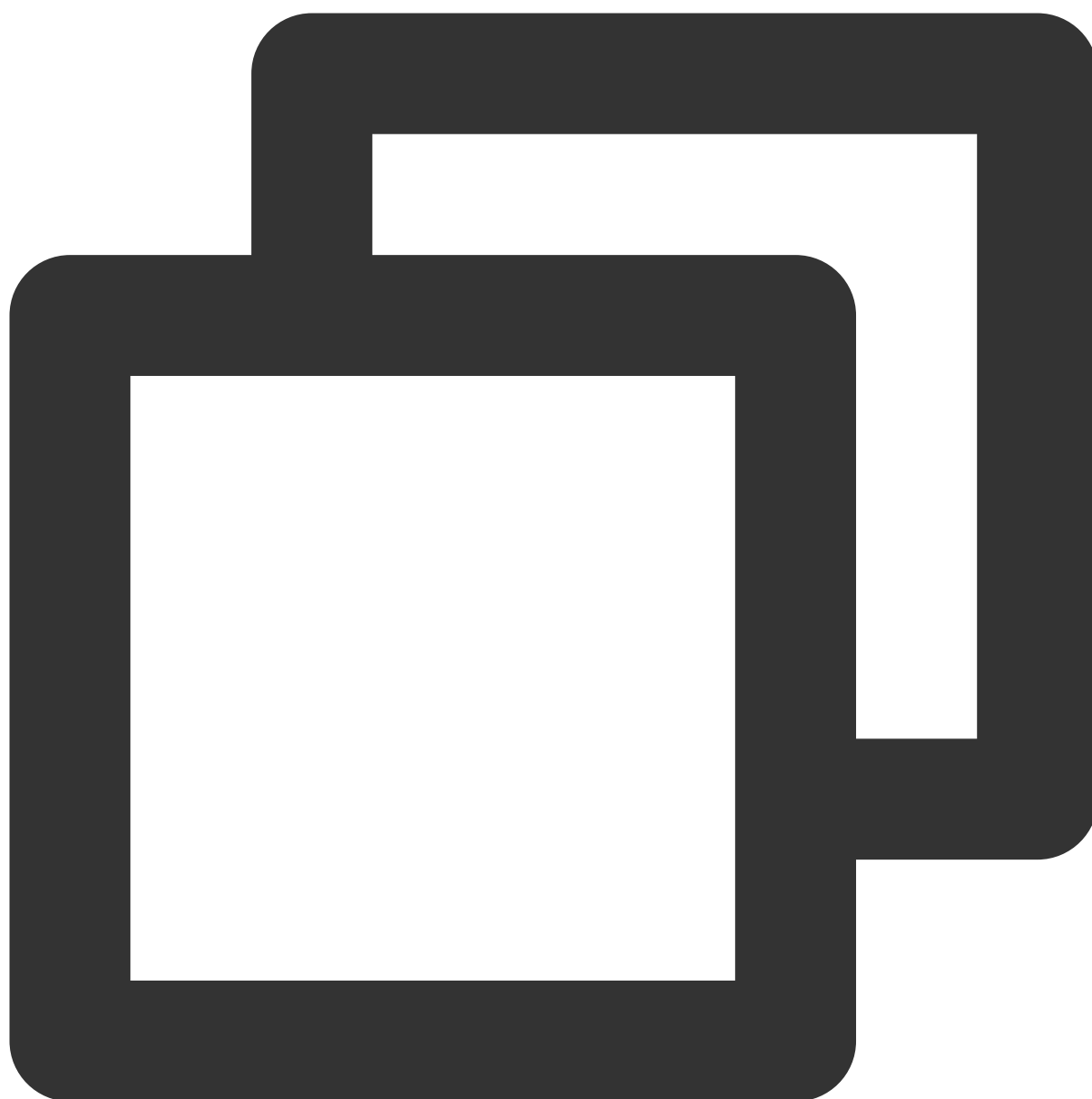
        //注意：requestCode必须>=1000000，否则可能与内部requestCode冲突，引起未知问题
        activity.startActivityForResult(new Intent(activity, TransActivity.class), 1000000);
    }
}
```

在自定义 API 中调用三方 App

当在自定义 API 中调用其它第三方 App（分享、支付、登录）等，当这些操作完成后能直接返回小程序而不是返回 App。

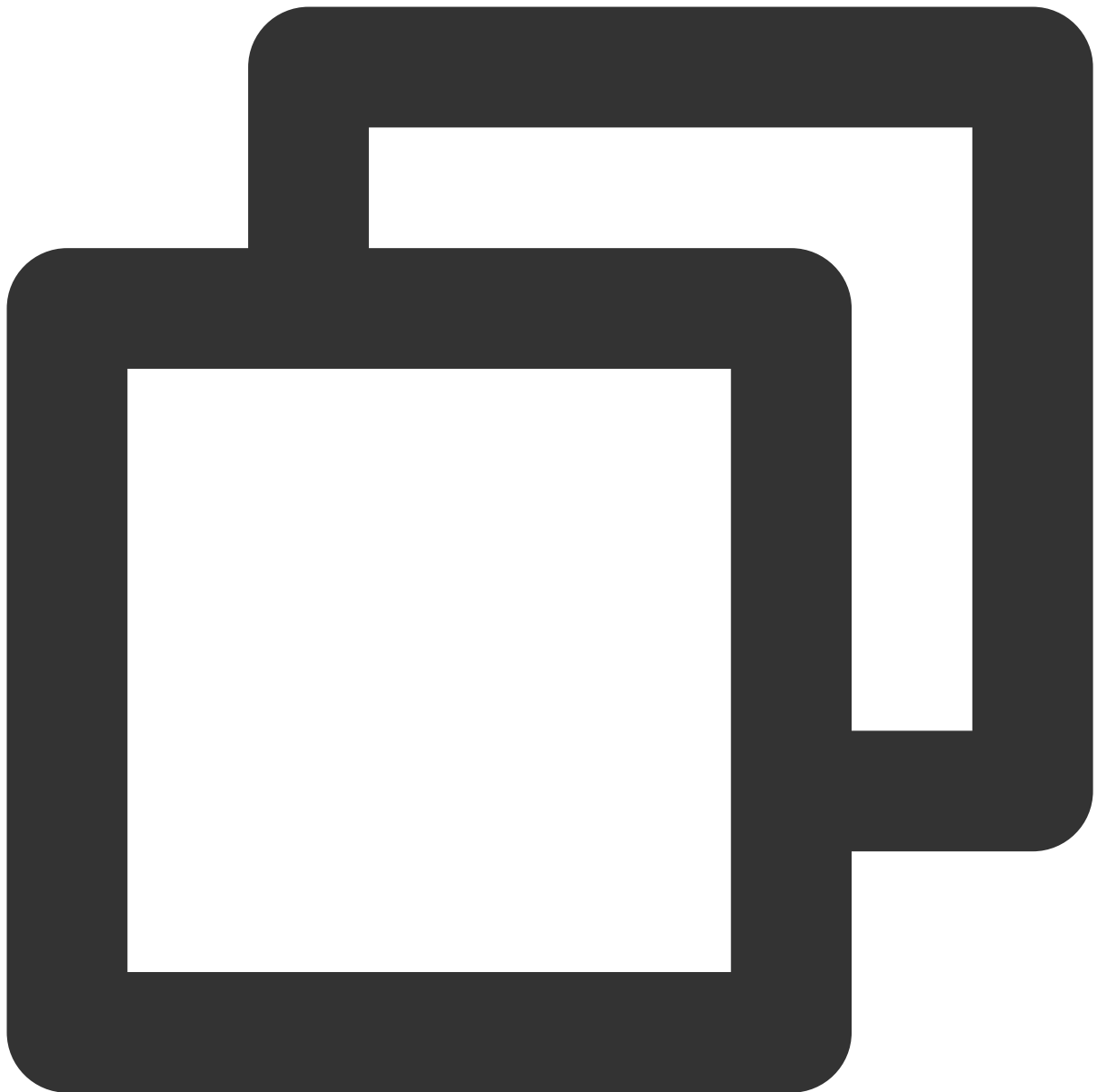
首先开发者创建透明的辅助 Activity，如下示例：

TransActivity 为定义的透明辅助 activity（透明 activity 开发者自行配置，因为主题的设置与应用 Activity 继承的系统 Activity 有关）



```
<activity android:name=".activity.TransActivity"
    android:exported="false"
    android:screenOrientation="portrait"
    android:theme="@style/TransparentTheme"/>
```

然后在 `TransActivity` 中调用业务逻辑。



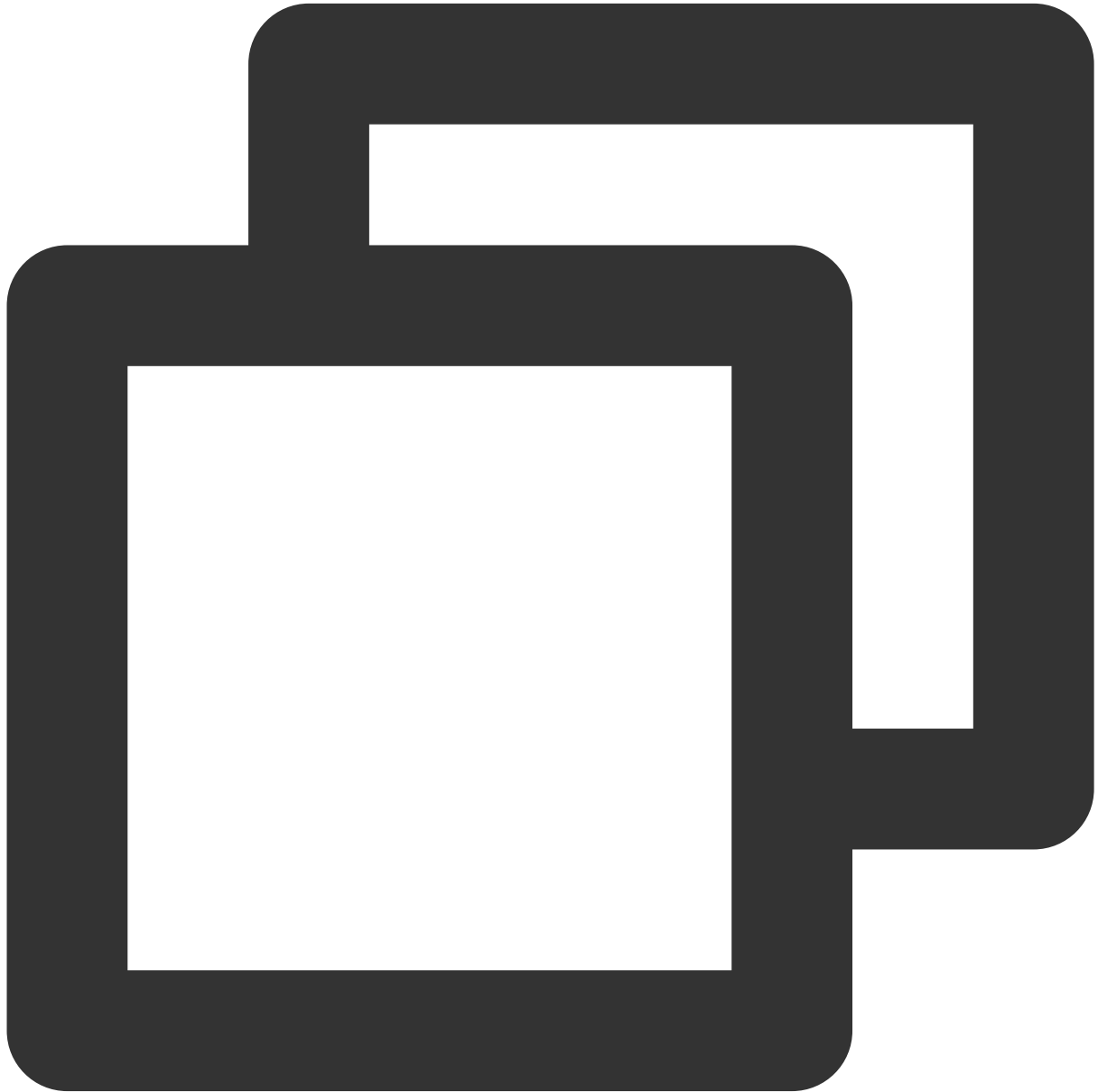
```
public class TransActivity extends AppCompatActivity {
    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.applet_activity_tran);

        /*****
            业务开发
        *****/

        //业务开发完成后返回数据并调用finish
    }
}
```

```
Intent intent = new Intent();
intent.putExtra("key", "value");
setResult(RESULT_OK, intent);
finish();
}
}
```

插件启动 **Activity** 并监听数据返回。



```
@JsPlugin(secondary = true)
public class CustomPlugin extends BaseJsPlugin {
    @JsEvent("testStartActivityForResult")
```

```
public void testStartActivityResult(final RequestEvent req) {
    Activity activity = req.activityRef.get();
    TmfMiniSDK.addActivityResultListener(new IActivityResultListener() {
        @Override
        public boolean doOnActivityResult(int requestCode, int resultCode, Intent data) {
            TmfMiniSDK.removeActivityResultListener(this);

            Log.i(ModuleApplet.TAG, data.getStringExtra("key"));
            req.ok();
            return true;
        }
    });

    //注意：requestCode必须>=1000000，否则可能与内部requestCode冲突，引起未知问题
    activity.startActivityForResult(new Intent(activity, TransActivity.class), 1000000);
}
```

自定义小程序视图组件

最近更新时间：2024-06-27 10:55:35

小程序视图组件扩展说明

小程序 SDK 的使用者可以自定义客户端原生组件为小程序提供支持。小程序开发者通过特定的小程序 API 能够在小程序页面中创建并操作原生组件以及和原生组件进行通信。Android 的客户端原生组件分为两种类型：

非同层原生组件：原生组件渲染在小程序页面之上，不支持 `zIndex`，始终位于小程序其他组件上层并会遮挡小程序内容。该类型组件以普通 `View` 的形式进行绘制。

同层原生组件：原生组件渲染在小程序页面内，支持 `zIndex` 以及页面内的组件层级关系。该类型组件需要以 `Surface` 的形式绘制在小程序页面中。

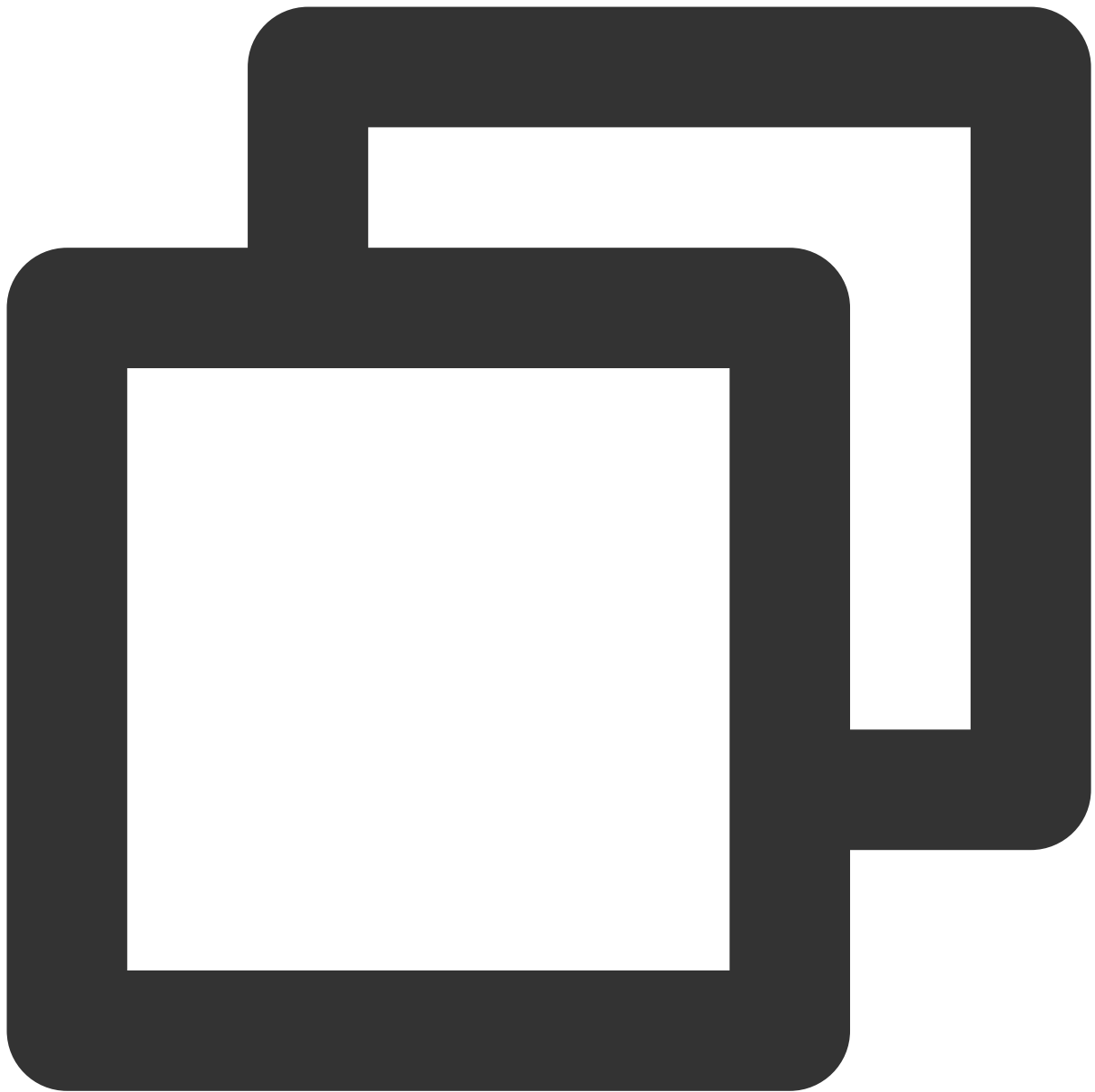
说明：
为了实现扩展组件，依赖于小程序开发者的在小程序中的改造和宿主应用开发者在宿主应用中的改造，二者结合才能实现小程序视图组件的扩展。

原生组件扩展实现

应用的开发者需要实现特定的代理，在小程序调用自定义组件 API 时创建原生组件以及响应对原生组件的操作。以下会针对非同层组件以同层组件分别进行说明：

非同层原生组件扩展

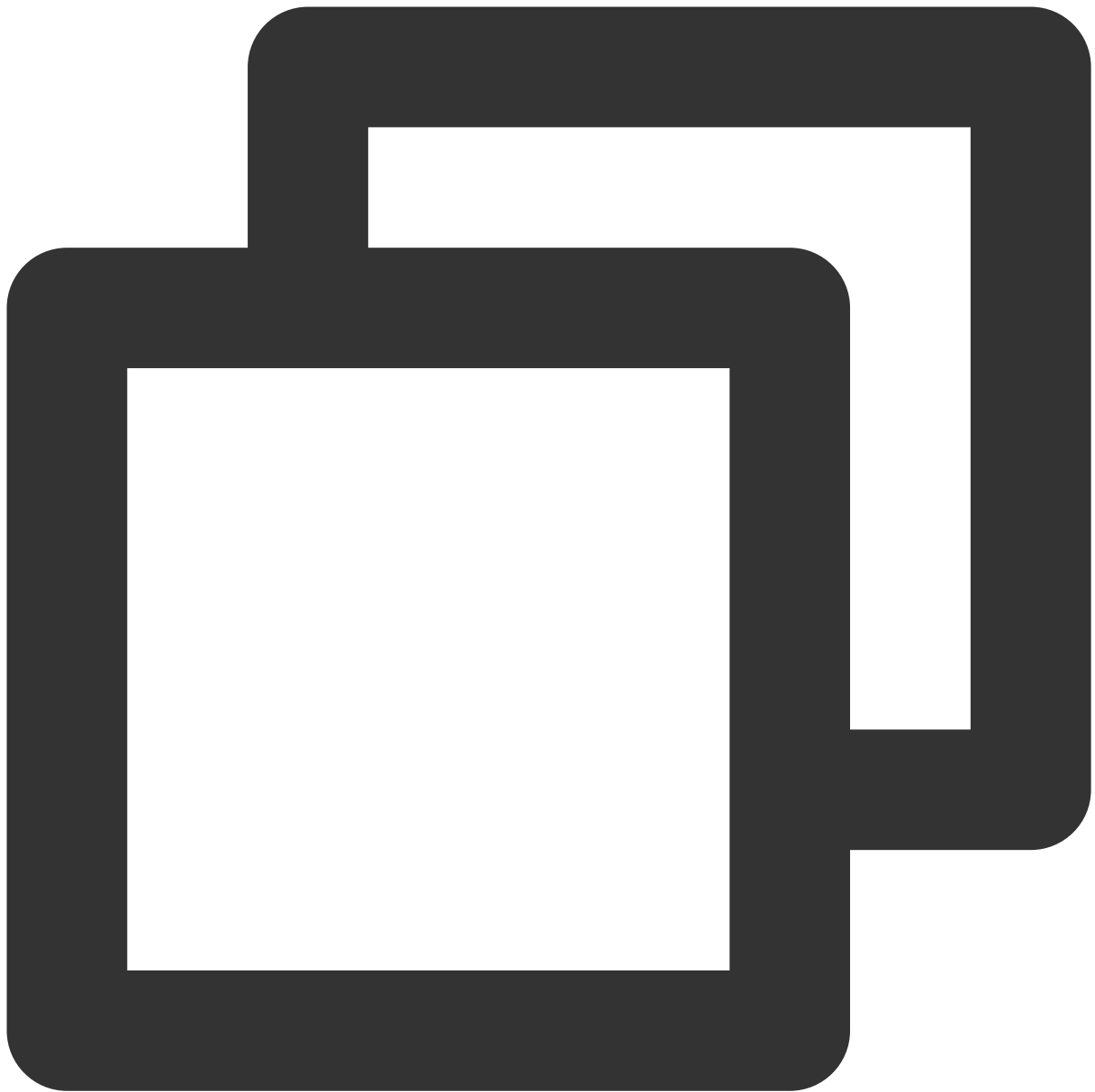
通过如下设置覆写代理，宿主可以自定义创建非同层原生组件：



```
@ProxyService(proxy = ExternalElementProxy.class)
public class MyExternalElementProxy extends ExternalElementProxy{}
```

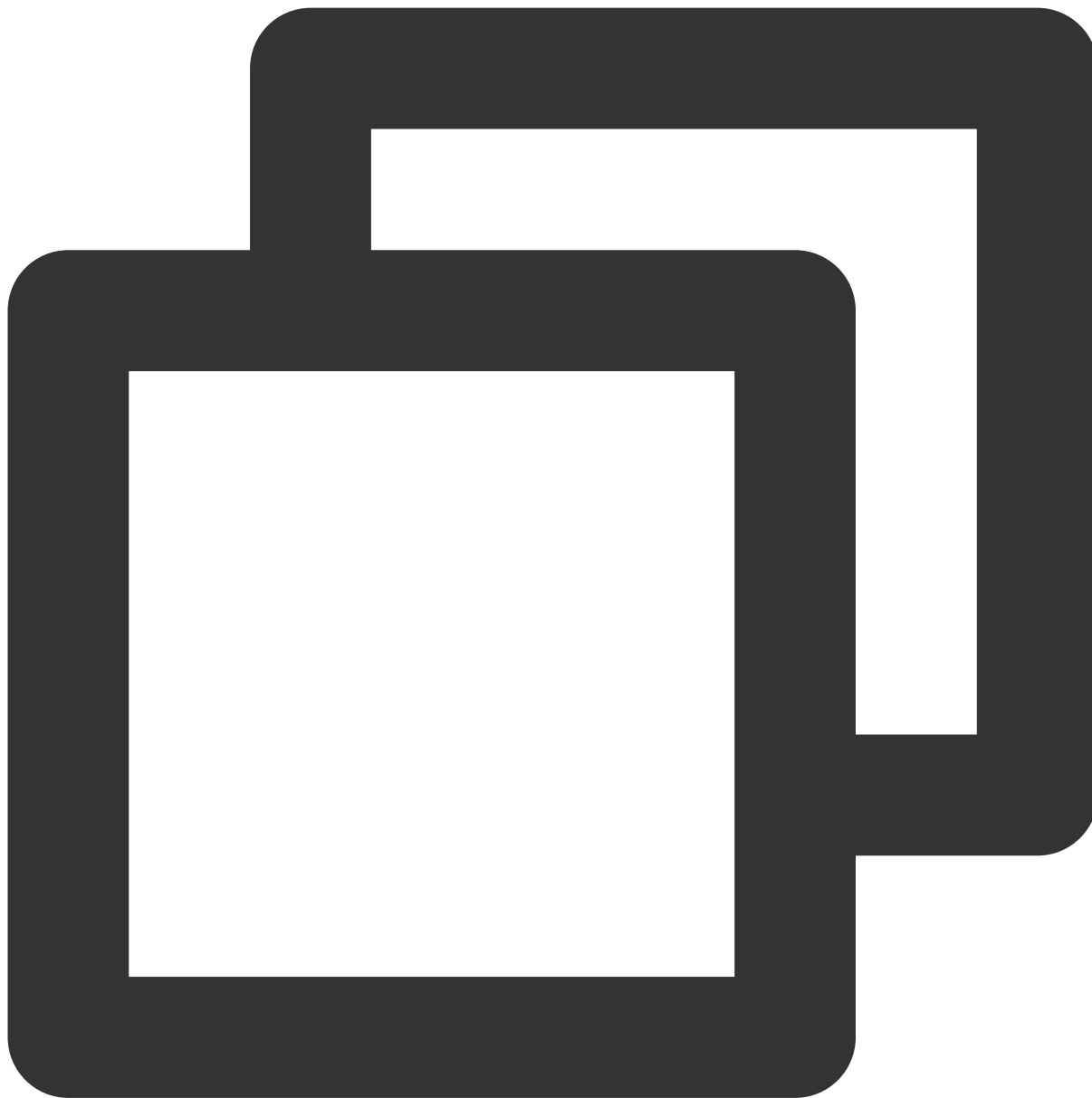
代理需要实现以下内容：

1. 插入非同层组件。当小程序向页面中插入非同层原生组件时会调用此方法。开发者需要实现此方法，并将自定义组件作为子 **View** 加入到 **parent** 参数提供的容器中。



```
/**
 * 创建非同层组件，当小程序创建自定义非同层原生组件时将会调用此接口
 *
 * @param widgetId 小程序创建的组件唯一ID
 * @param widgetContext 小程序组件的上下文，用于向小程序回传内容
 * @param type 小程序创建的组件类型名称
 * @param parent 承载原生组件的父容器
 * @param params 小程序创建组件时传递的参数
 */
public abstract void handleInsertElement(long widgetId, ExternalWidgetContext widgetContext,
String type, ViewGroup parent, JSONObject params);
```

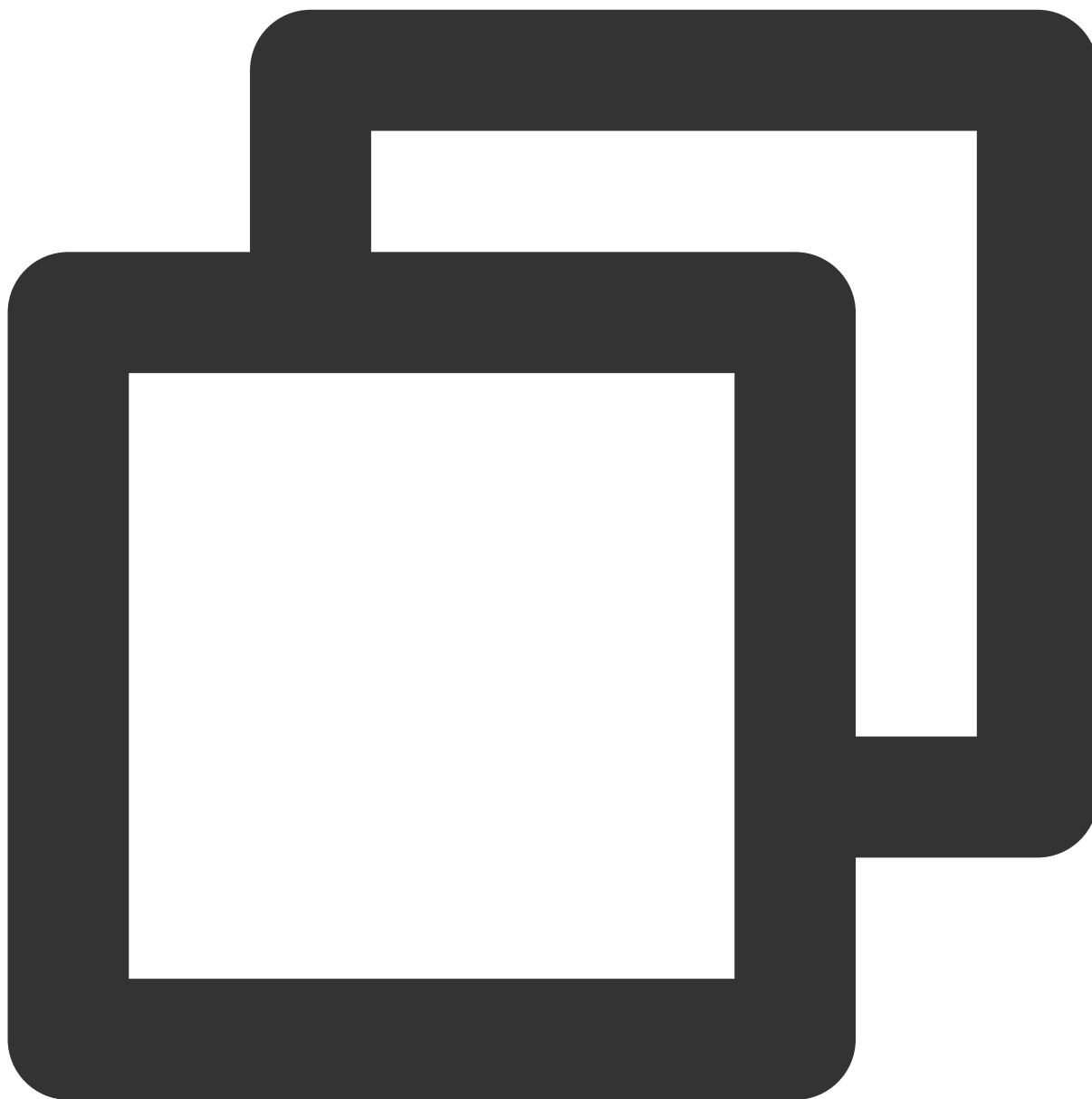
2. 更新非同层组件。当原生组件在小程序中的位置以及大小发生变化时会调用此方法通知应用。原生组件的父容器布局会按照变化后的样式进行自适应，原生组件本身则可以根据需要对自身进行调整。



```
/**
 * 更新非同层组件样式，当小程序更新自定义非同层原生组件的样式时将会调用此接口
 *
 * @param widgetId 小程序组件ID
 * @param widgetContext 小程序组件的上下文，用于向小程序回传内容
 * @param params 小程序更新组件时传递的参数
 */
```

```
public abstract void handleUpdateElement(long widgetId, ExternalWidgetContext widgetContext,
                                          JSONObject params);
```

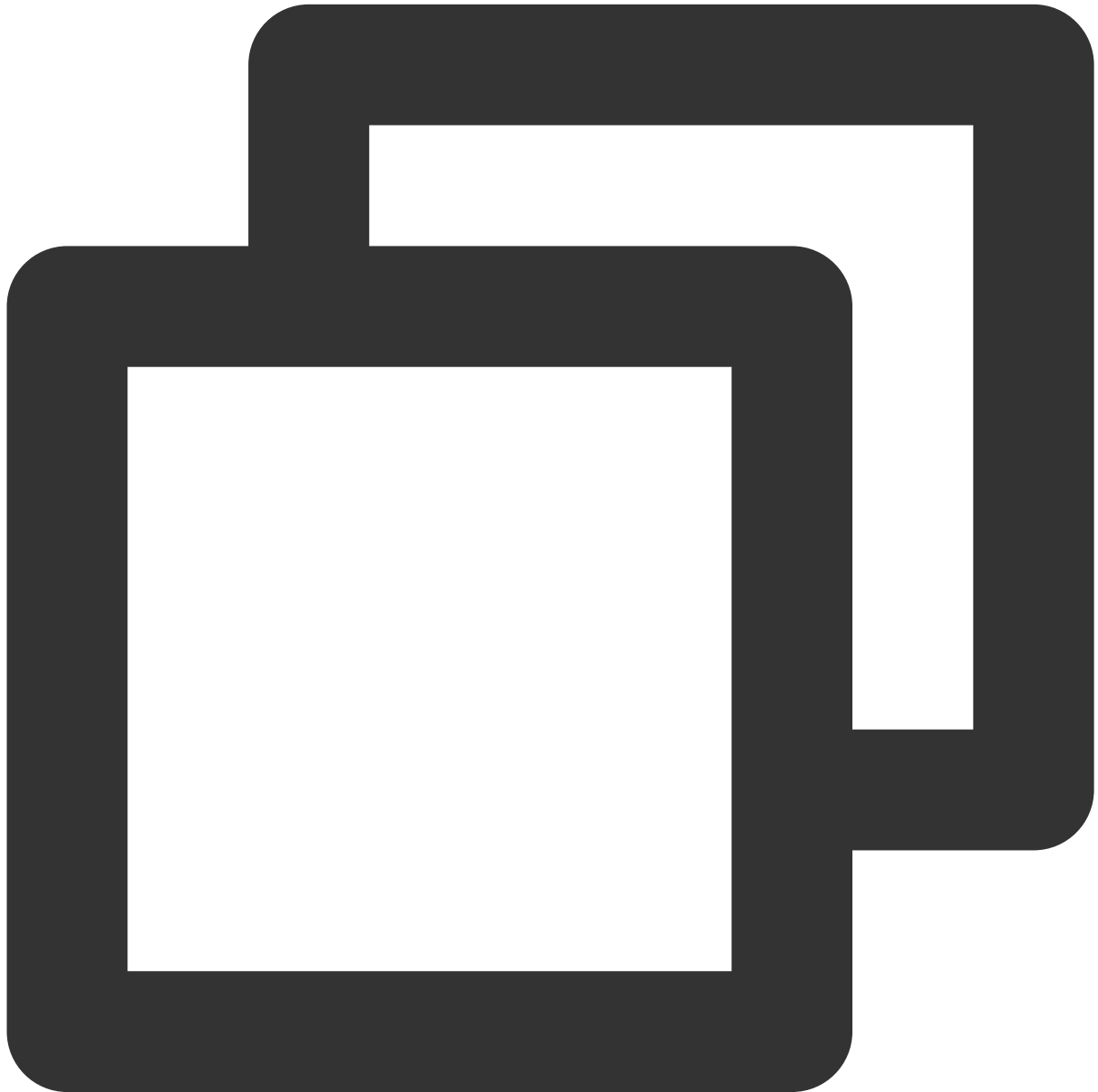
3. 操作非同层组件。当小程序向原生组件发送事件时会调用此方法（例如按钮点击，状态变更），事件的具体内容需要开发者自行在 `params` 中进行定义。



```
/**
 * 操作非同层组件，当小程序需要向非同层组件发送指令或者调用特有方法的时候将会调用此接口
 *
 * @param widgetId 小程序组件ID
 * @param widgetContext 小程序组件的上下文，用于向小程序回传内容
```

```
* @param params 小程序更新组件时传递的参数
*/
public abstract void handleOperateElement(long widgetId, ExternalWidgetContext widg
                                           JSONObject params);
```

4. 删除非同层组件。当小程序删除已添加的原生组件时将调用此方法通知应用。此时组件的父容器将被移除，应用应当对组件进行销毁处理。

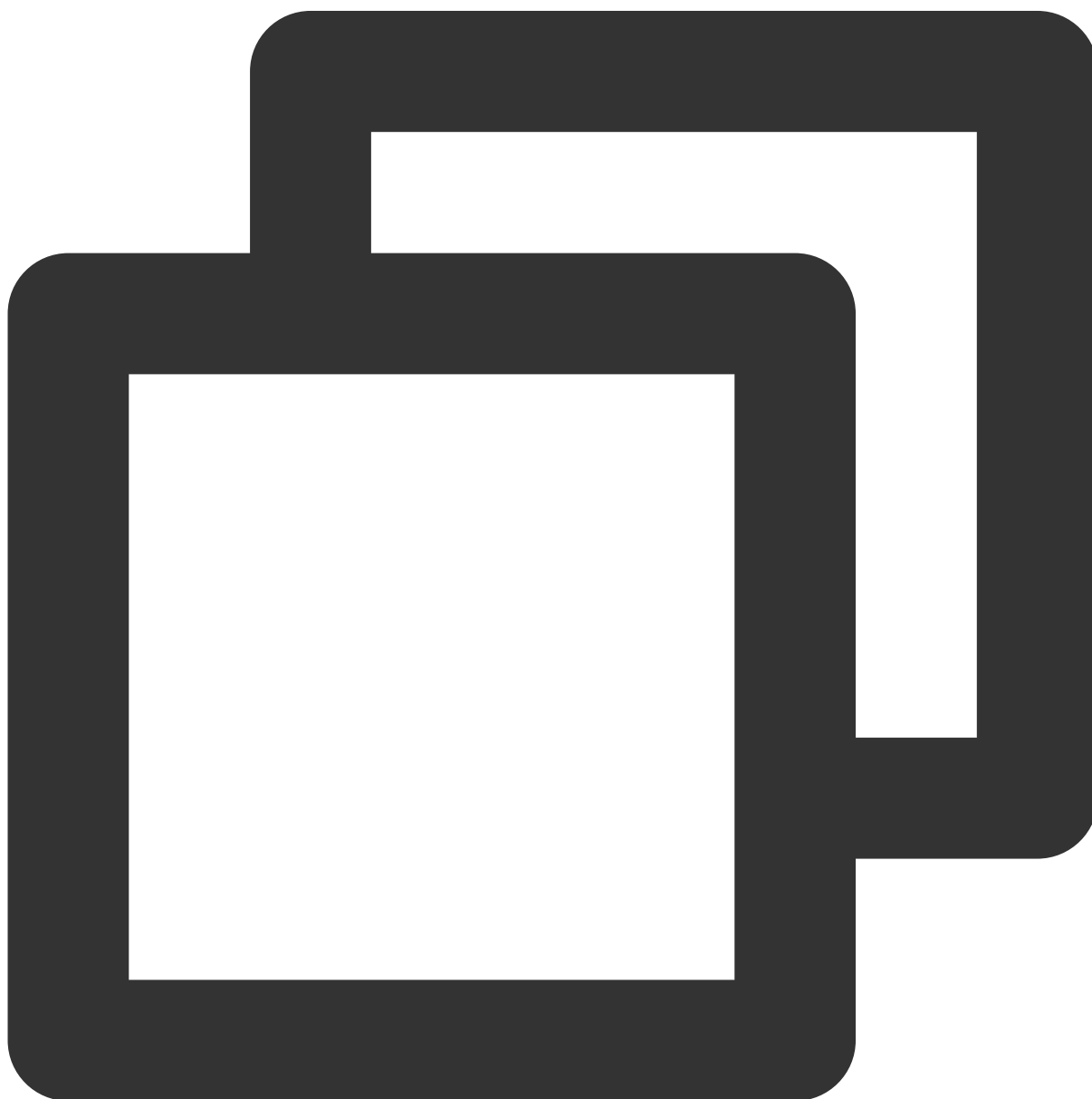


```
/**
 * 删除非同层组件
 */
```

```
* @param widgetId 小程序组件ID
* @param widgetContext 小程序组件的上下文，用于向小程序回传内容
*/
public abstract void handleRemoveElement(long widgetId, ExternalWidgetContext widge
```

小程序非同层原生组件上下文

小程序组件的上下文包含了使原生组件能够向对应的小程序组件发送消息的方法。`onExternalElementEvent` 方法会直接向小程序发送 `onExternalElementEvent` 事件，小程序应该捕获并处理该事件；`callbackSuccess`、`callbackFail` 则是在小程序调用 API 向应用发送事件时对当次小程序 API 调用传入的 `success` 或者 `fail` 方法进行回调。



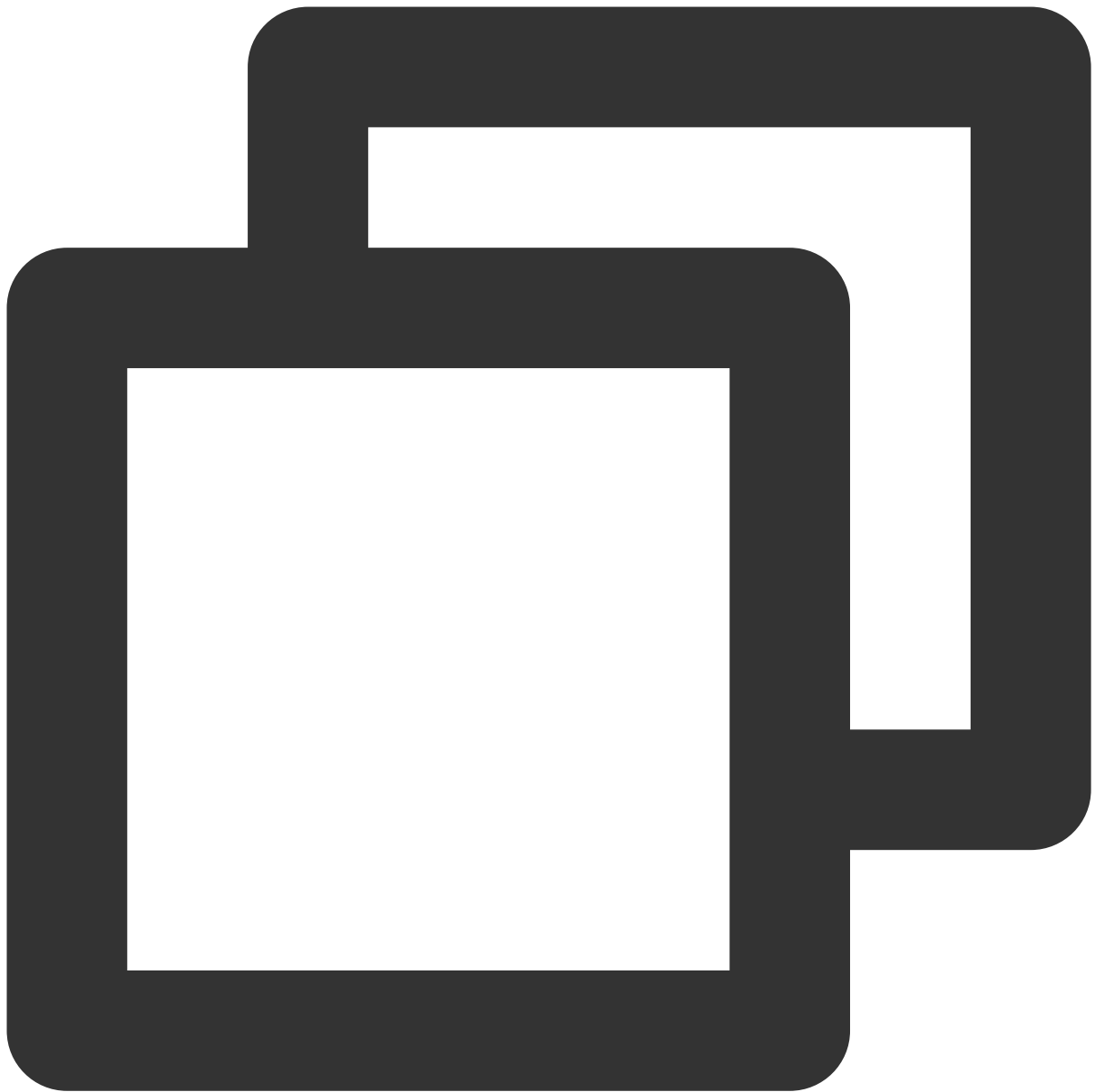
```
/**
 * 向小程序发送 onExternalElementEvent 事件
 *
 * @param jsonObject 事件携带的JSON数据
 */
public final void onExternalElementEvent(JSONObject jsonObject);

/**
 * 调用小程序提供的success回调方法
 *
 * @param jsonObject 回调携带的JSON数据，可以为空
 */
public final void callbackSuccess(JSONObject jsonObject);

/**
 * 调用小程序提供的fail回调方法
 *
 * @param jsonObject jsonObject 回调携带的JSON数据，可以为空
 * @param message 错误信息描述
 */
public final void callbackFail(JSONObject jsonObject, String message);
```

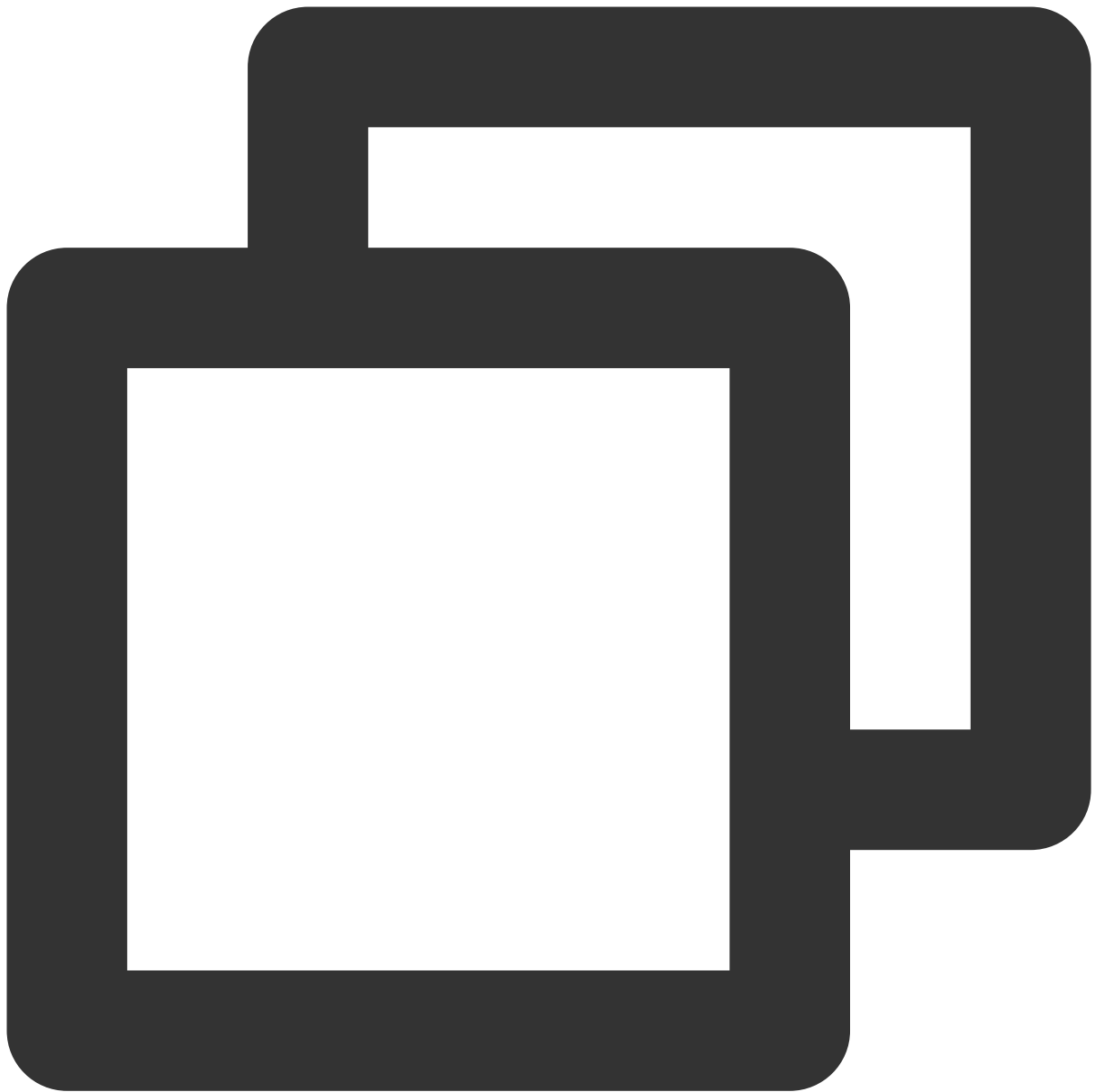
同层原生组件扩展

如果小程序请求创建的是同层原生组件，则需要应用实现以下代理：



```
@ProxyService(proxy = ExternalEmbeddedWidgetClient.class)
public class MyExternalEmbeddedElementProxy extends ExternalEmbeddedWidgetClient{}
```

1. 与非同层组件不同，同层组件由于涉及到嵌入页面的 **Surface** 创建以及销毁，因此会提供额外的组件生命周期回调。开发者可以覆写这些方法对同层组件的生命周期事件进行处理：



```
/**
 * 初始化同层组件DOM节点
 *
 * @param context 页面上下文
 * @param widgetId 同层组件ID
 * @param tagName 同层组件在DOM中的TAG
 * @param attributes 同层组件的Attributes
 */
public abstract void onInit(Context context, long widgetId, String tagName, Map<Str

/**
```



```
* 同层组件Surface创建完毕回调
*
* @param widgetId 同层组件ID
* @param surface 同层组件关联的Surface
*/
public abstract void onSurfaceCreated(long widgetId, Surface surface);

/**
 * 同层组件Surface销毁回调
 *
 * @param widgetId 同层组件ID
 * @param surface 同层组件关联的Surface
 */
public abstract void onSurfaceDestroyed(long widgetId, Surface surface);

/**
 * 同层组件触摸事件回调
 *
 * @param widgetId 同层组件ID
 * @param event 触摸事件
 * @return 同层组件是否消费触摸事件
 */
public abstract boolean onTouchEvent(long widgetId, MotionEvent event);

/**
 * 同层组件绘制区域变化回调
 *
 * @param widgetId 同层组件ID
 * @param rect 新的绘制区域
 */
public abstract void onRectChanged(long widgetId, Rect rect);

/**
 * 同层组件请求重绘
 *
 * @param widgetId 同层组件ID
 */
public abstract void onRequestRedraw(long widgetId);

/**
 * 同层组件可见性发生变化
 *
 * @param widgetId 同层组件ID
 * @param visibility 是否可见
 */
public abstract void onVisibilityChanged(long widgetId, boolean visibility);
```

```
/**
 * 同层组件变为可用
 *
 * @param widgetId 同层组件ID
 */
public abstract void onActive(long widgetId);

/**
 * 同层组件变为不可用
 *
 * @param widgetId 同层组件ID
 */
public abstract void onDeActive(long widgetId);

/**
 * 同层组件节点被销毁
 *
 * @param widgetId 同层组件ID
 */
public abstract void onDestroy(long widgetId);

/**
 * 当前小程序页面进入pause状态
 *
 * @param widgetId 同层组件ID
 */
public abstract void webViewPause(long widgetId);

/**
 * 当前小程序页面进入resume状态
 *
 * @param widgetId 同层组件ID
 */
public abstract void webViewResume(long widgetId);

/**
 * 当前小程序页面被销毁
 *
 * @param widgetId 同层组件ID
 */
public abstract void webViewDestroy(long widgetId);

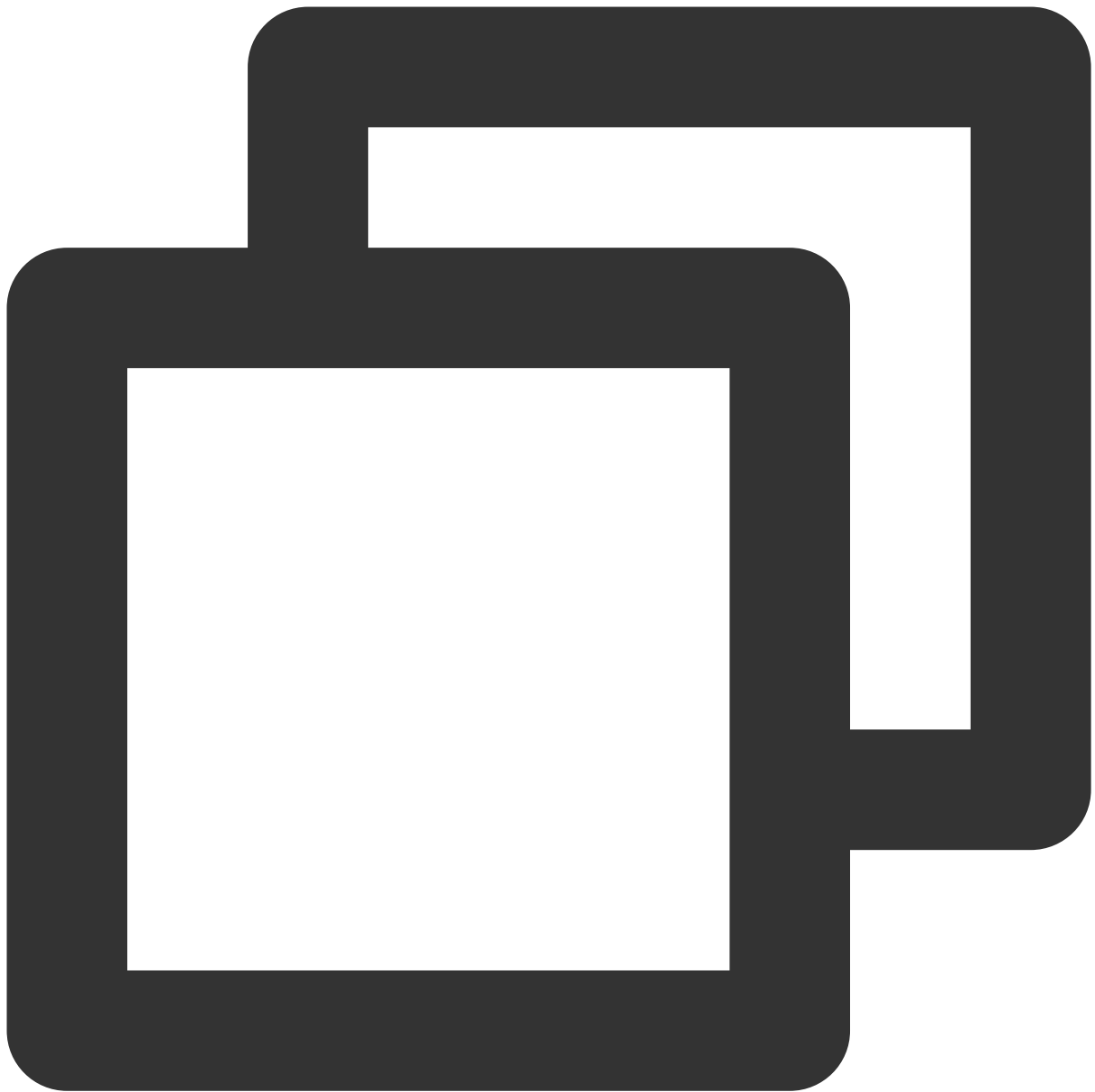
/**
 * 小程序进入resume状态
 *
 * @param widgetId 同层组件ID
 */
```

```
public abstract void nativeResume(long widgetId);

/**
 * 小程序进入pause状态
 *
 * @param widgetId 同层组件ID
 */
public abstract void nativePause(long widgetId);

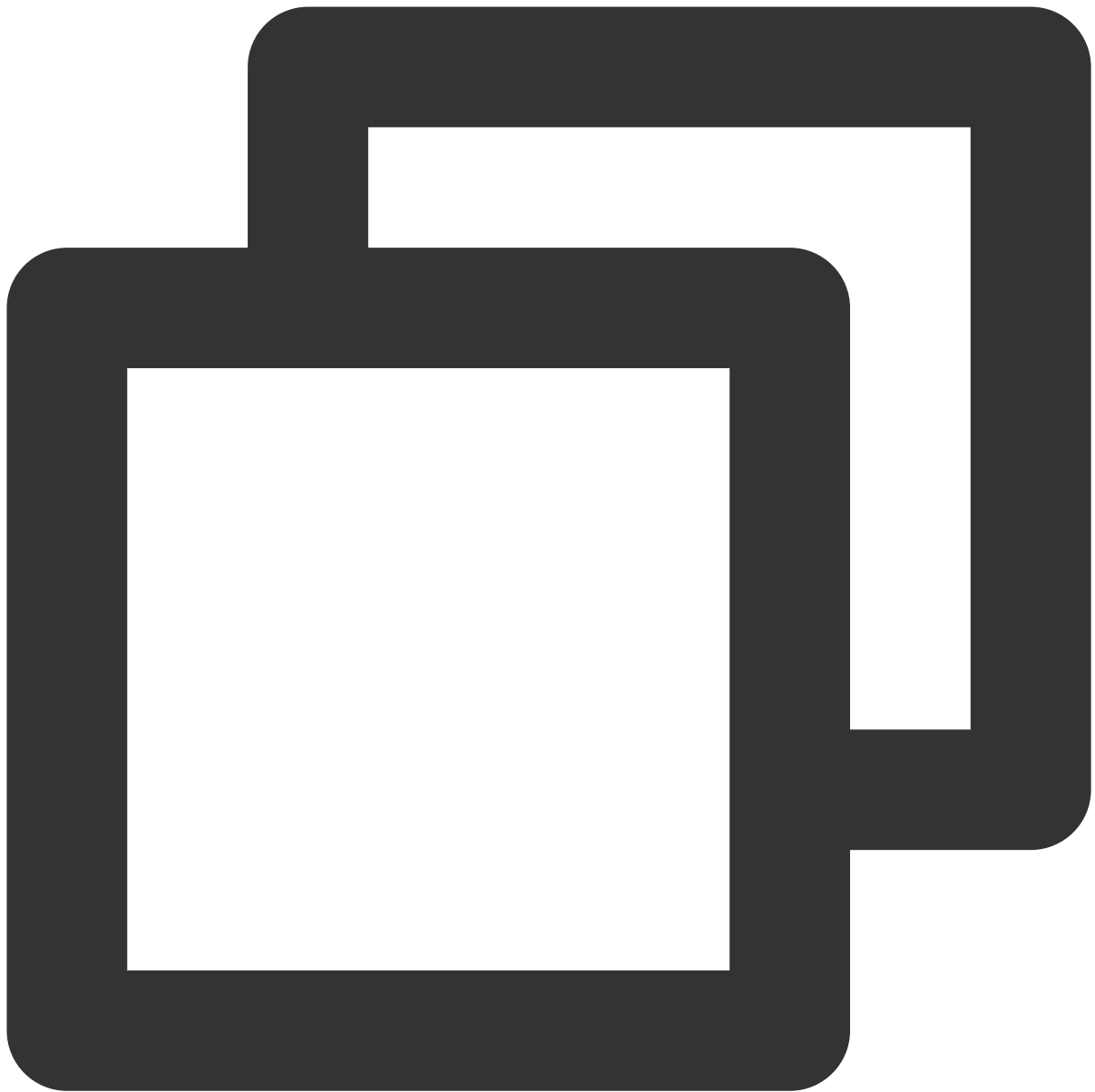
/**
 * 小程序被销毁
 *
 * @param widgetId 同层组件ID
 */
public abstract void nativeDestroy(long widgetId);
```

2. 当小程序真正向页面插入同层组件时会调用应用的 `handleInsertXWebExternalElement` 接口。客户端开发应当实现该方法，根据 `type` 提供的组件类型将相应的组件绘制到 `widgetId` 关联的 `Surface` 上。



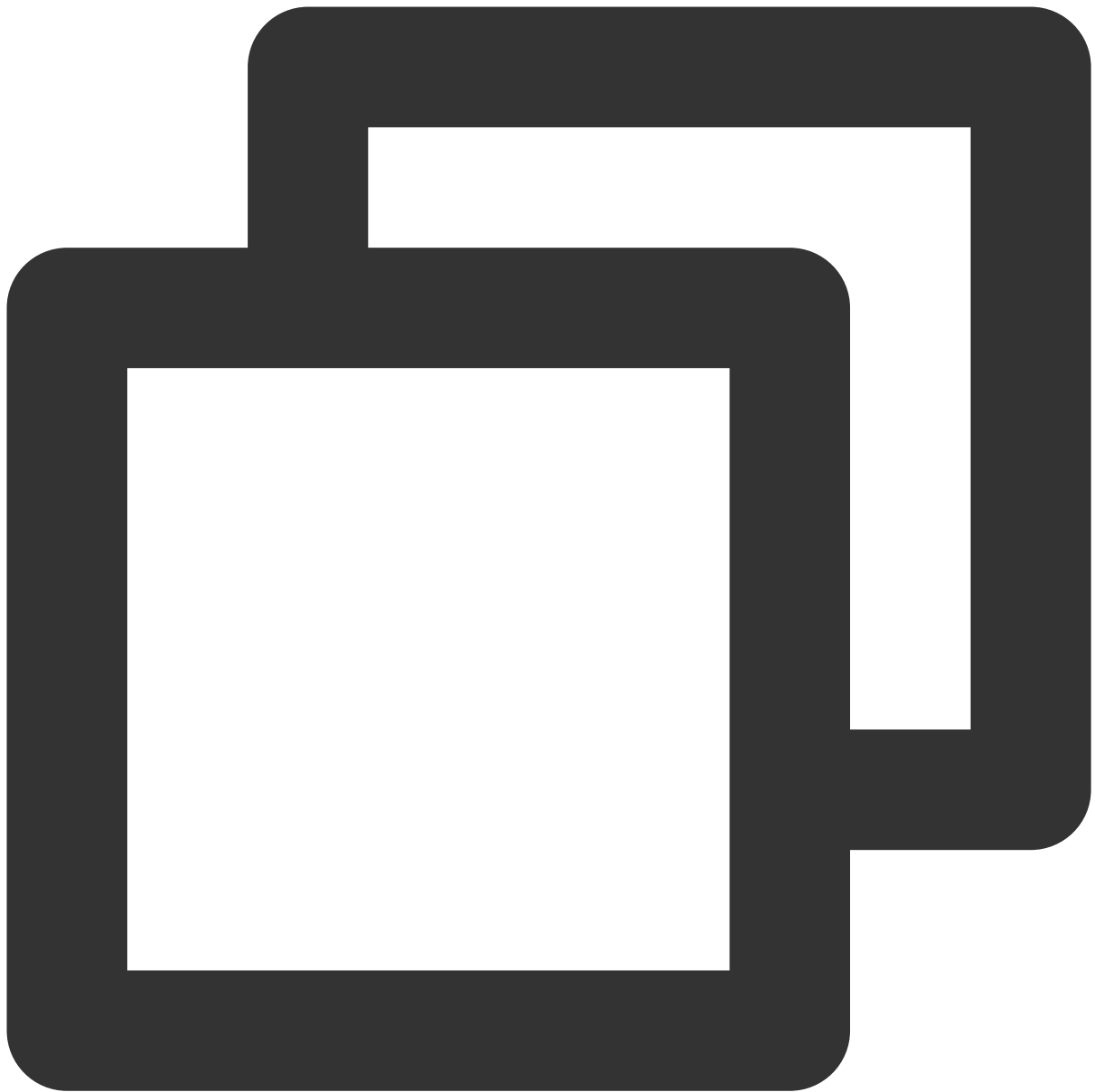
```
/**
 * 插入同层组件
 *
 * @param widgetId 同层组件ID
 * @param widgetContext 同层组件关联的小程序上下文
 * @param type 同层组件类型
 * @param req 同层组件透传参数
 */
public abstract void handleInsertXWebExternalElement(long widgetId, ExternalWidgetC
```

3. 组件的大小以及样式发生变化时，会通过该方法通知应用：



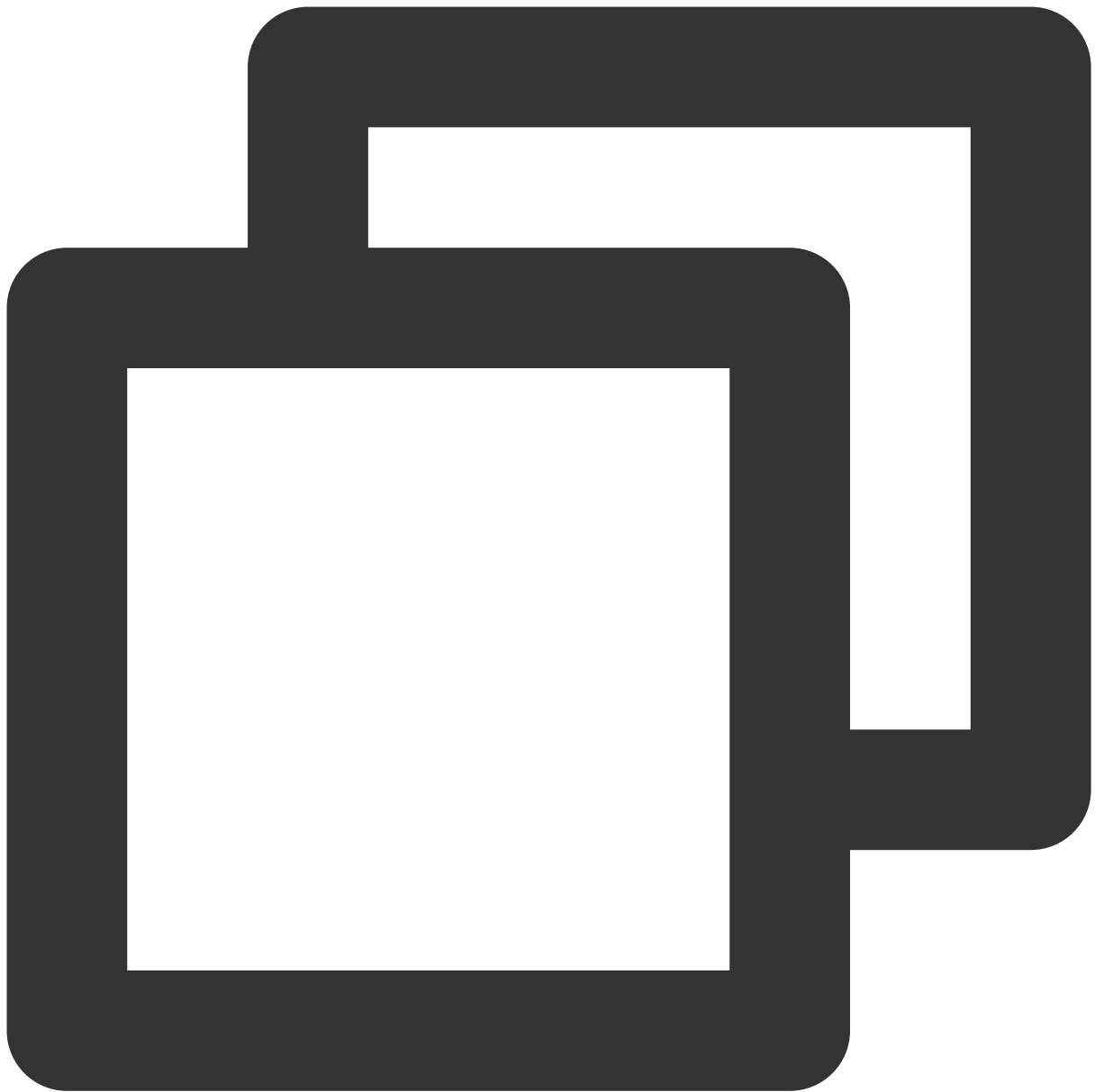
```
/**
 * 更新同层组件样式
 *
 * @param widgetId 同层组件ID
 * @param widgetContext 同层组件关联的小程序上下文
 * @param req 小程序传递的参数
 */
public abstract void handleUpdateXWebExternalElement(long widgetId, XWebExternalWid
```

4. 当小程序组件向原生组件发送事件需要操作原生组件时会通过该方法通知应用：



```
/**
 * 操作同层组件
 *
 * @param widgetId 同层组件ID
 * @param widgetContext 同层组件关联的小程序上下文
 * @param req 小程序传递的参数
 */
public abstract void handleOperateXWebExternalElement(long widgetId, XWebExternalWi
```

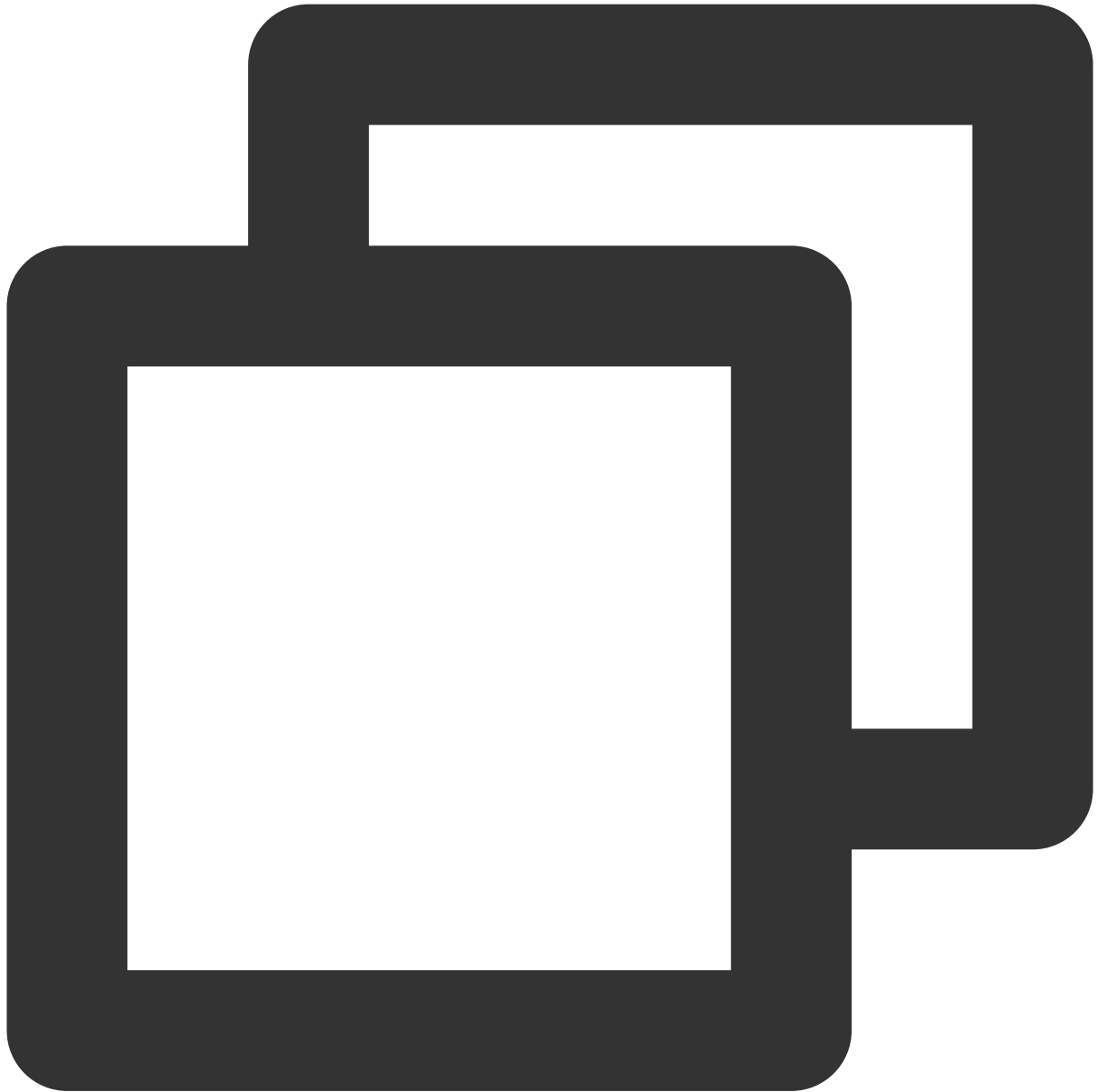
5. 当小程序删除同层原生组件时会通过该方法通知应用：



```
/**
 * 移除同层组件
 *
 * @param widgetId 同层组件ID
 * @param widgetContext 同层组件关联的小程序上下文
 */
public abstract void handleRemoveXWebExternalElement(long widgetId, XWebExternalWid
```

小程序同层原生组件上下文

与非同层组件类似，同层组件可以通过小程序组件的上下文向小程序发送 `onXWebExternalElementEvent` 事件；或是在小程序调用操作原生组件的 API 时回调 API 提供的 `callback` 方法。



```
/**
 * 向小程序发送 onXWebExternalElementEvent 事件
 *
 * @param jsonObject 事件携带的JSON数据
 */
public final void onXWebExternalElementEvent(JSONObject jsonObject);

/**
```

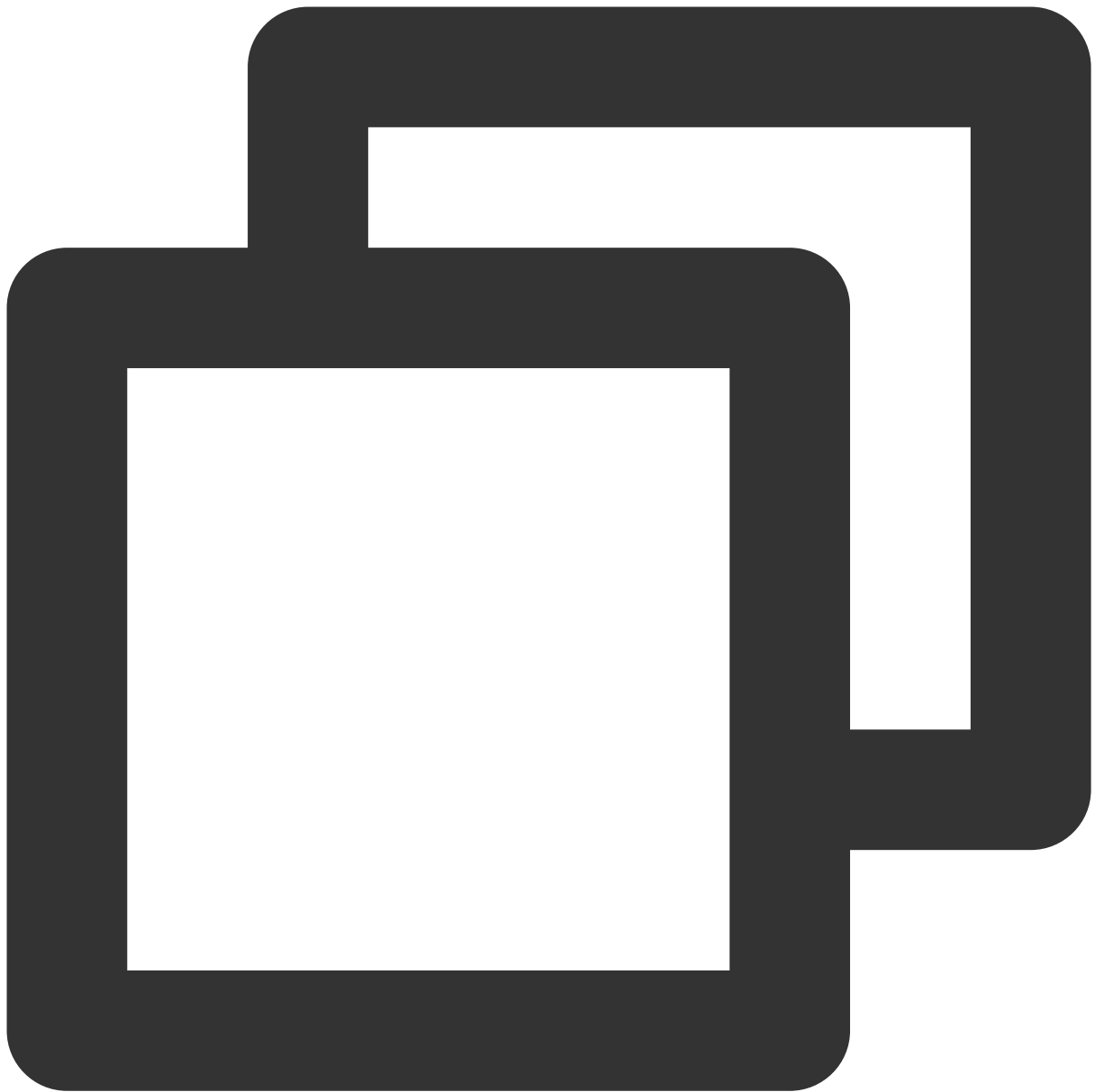


```
* 调用小程序提供的success回调方法
*
* @param jsonObject 回调携带的JSON数据，可以为空
*/
public final void callbackSuccess(JSONObject jsonObject);

/**
* 调用小程序提供的fail回调方法
*
* @param jsonObject jsonObject 回调携带的JSON数据，可以为空
* @param message 错误信息描述
*/
public final void callbackFail(JSONObject jsonObject, String message);
```

小程序扩展实现

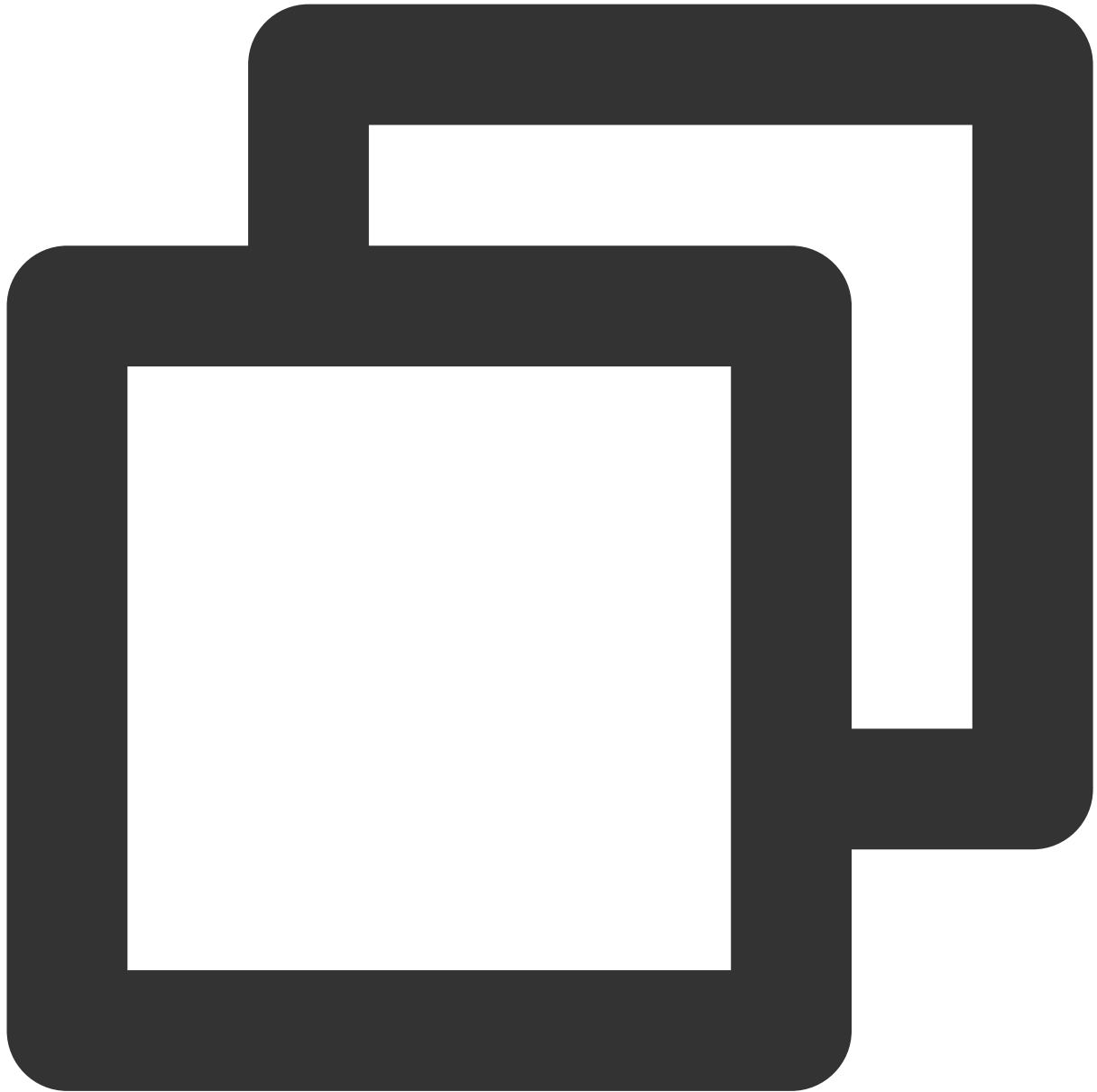
小程序向页面中插入客户端自定义的组件需要先在 `wxml` 中引入一个 `external-element` 节点：



```
<external-element
  id="comp1"
  type="maTestView"
  _insert2WebLayer="true"
  style="width: 200px;height: 100px;"
  bindexternalelementevent="handleEvent"
></external-element>
```

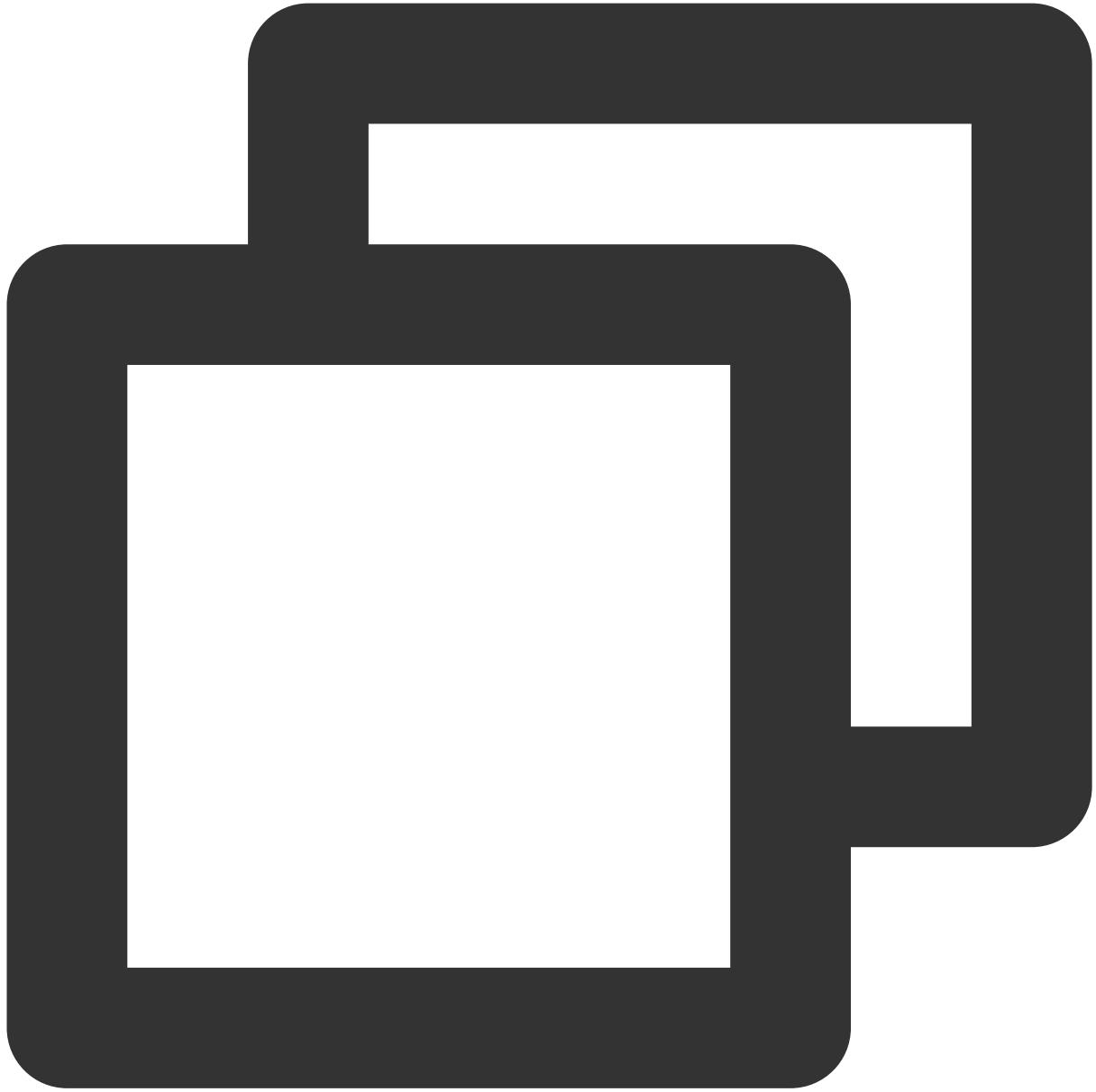
此处 `type` 为与应用约定创建的组件类型名称一致，`_insert2WebLayer` 表示该组件为同层组件或者非同层组件（`true` 为同层，需客户端实现同层组件代理；`false` 为非同层，需要客户端实现非同层代理），

`bindexternalelementevent` 可捕获 native 传递的 `onExternalElementEvent` 或者 `onXWebExternalElementEvent`，回调参数包括：



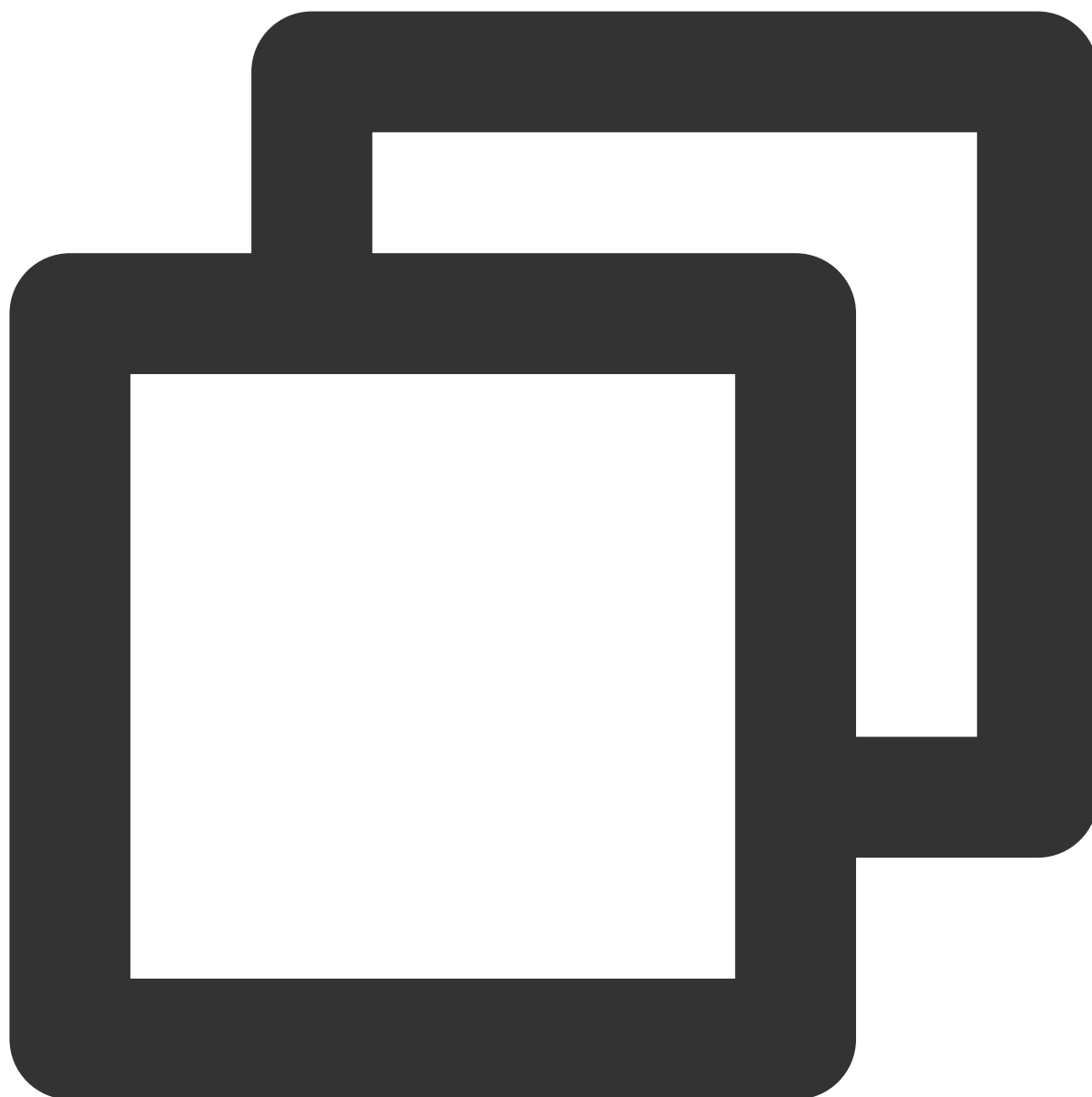
```
{
  target,
  currentTarget,
  timeStamp,
  touches,
  detail, // native传递的参数
}
```

之后小程序通过 id 创建一个与该节点相关联的上下文：



```
this.ctx = wx.createExternalElementContext('comp1');
```

该方法会通知应用在节点的位置创建对应的原生组件。小程序后续可以通过该上下文向原生组件发送事件，对原生组件进行操作：



```
this.ctx.call({
  params1: {
    name: 'name1',
    age: 11
  },
  params2: {
    name: 'name2',
    age: 22
  },
  success: (e) => {
    console.log('===operate success===', e)
```

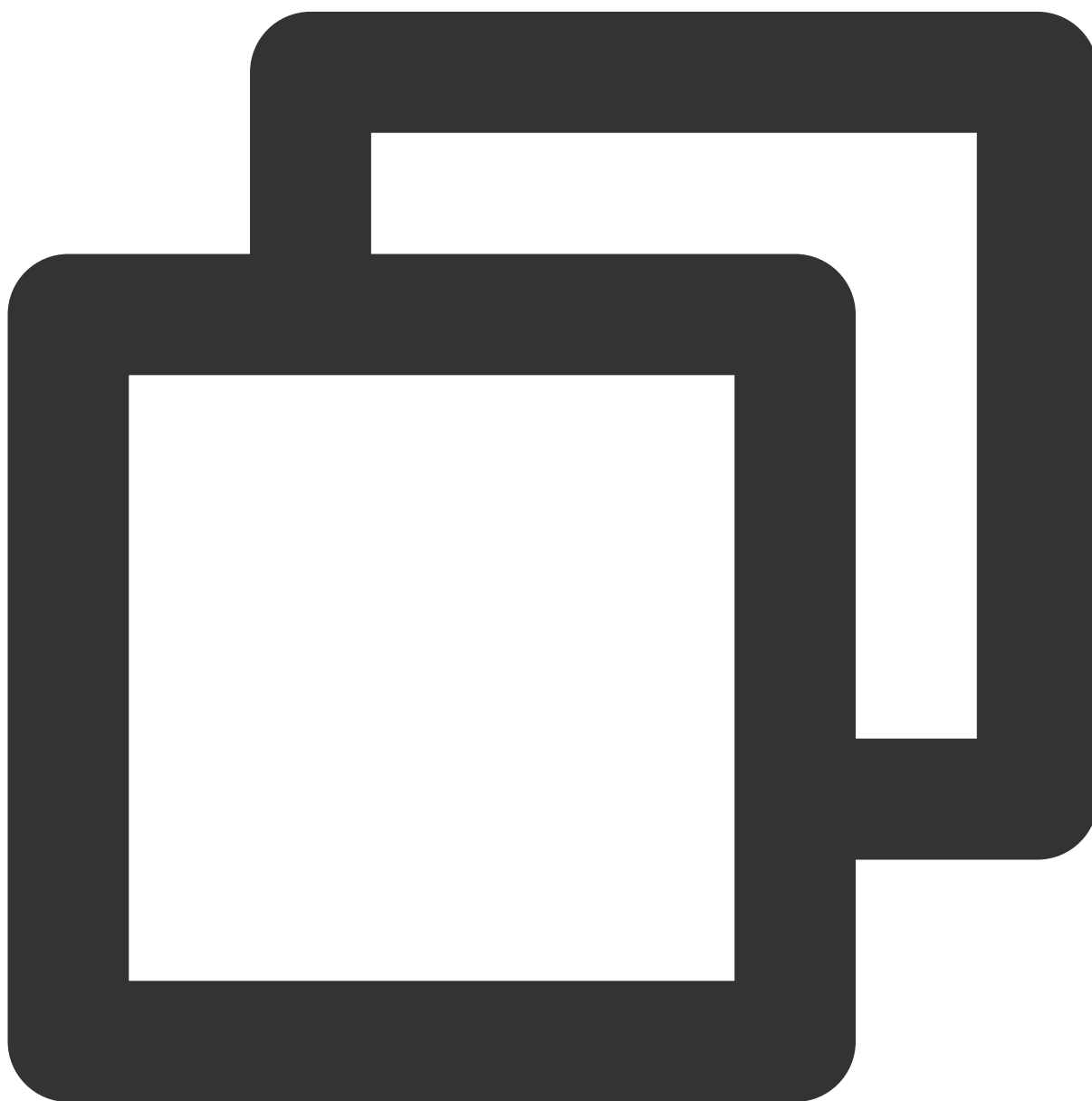
```
    },  
    fail: (e) => {  
      console.log('====operate fail====', e)  
    },  
    complete: (e) => {  
      console.log('====operate complete====', e)  
    }  
  })  
})
```

定制 SDK 能力

定制小程序 UI

最近更新时间：2024-06-27 10:55:04

小程序 SDK 开放了对小程序的部分原生 UI 效果的自定义能力，开发者可以按照下面的规范进行配置。
为了实现自定义小程序 UI，开发者需要继承并实现 `AbsMiniUiProxy`。示例代码如下：



```
@ProxyService(proxy = IMiniUiProxy.class)
```

```
public class MiniUiProxyImpl extends AbsMiniUiProxy
```

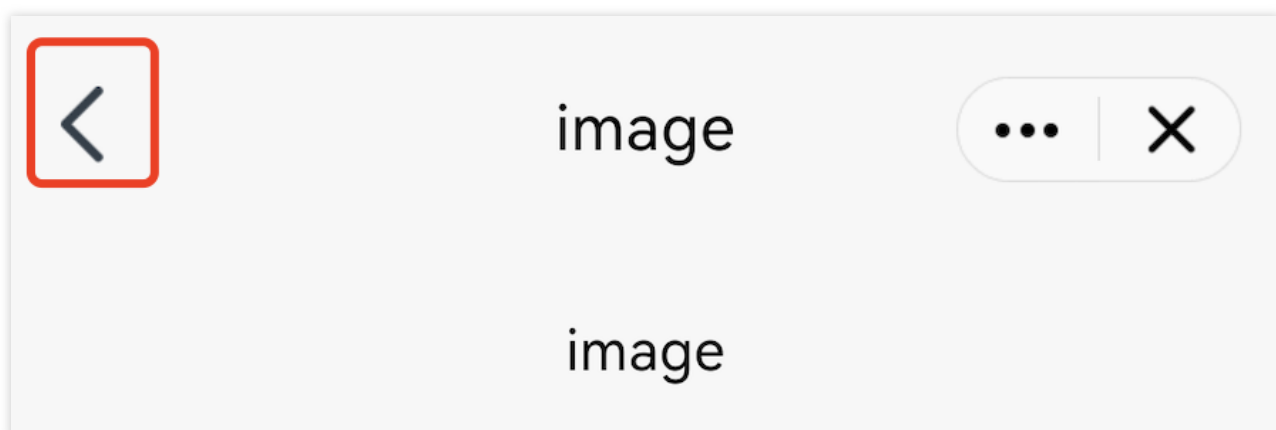
注意：

必须使用 `@ProxyService(proxy = IMiniUiProxy.class)` 注解自定义UI。

定制导航栏按钮

1. 返回按钮

返回按钮示意图如下：

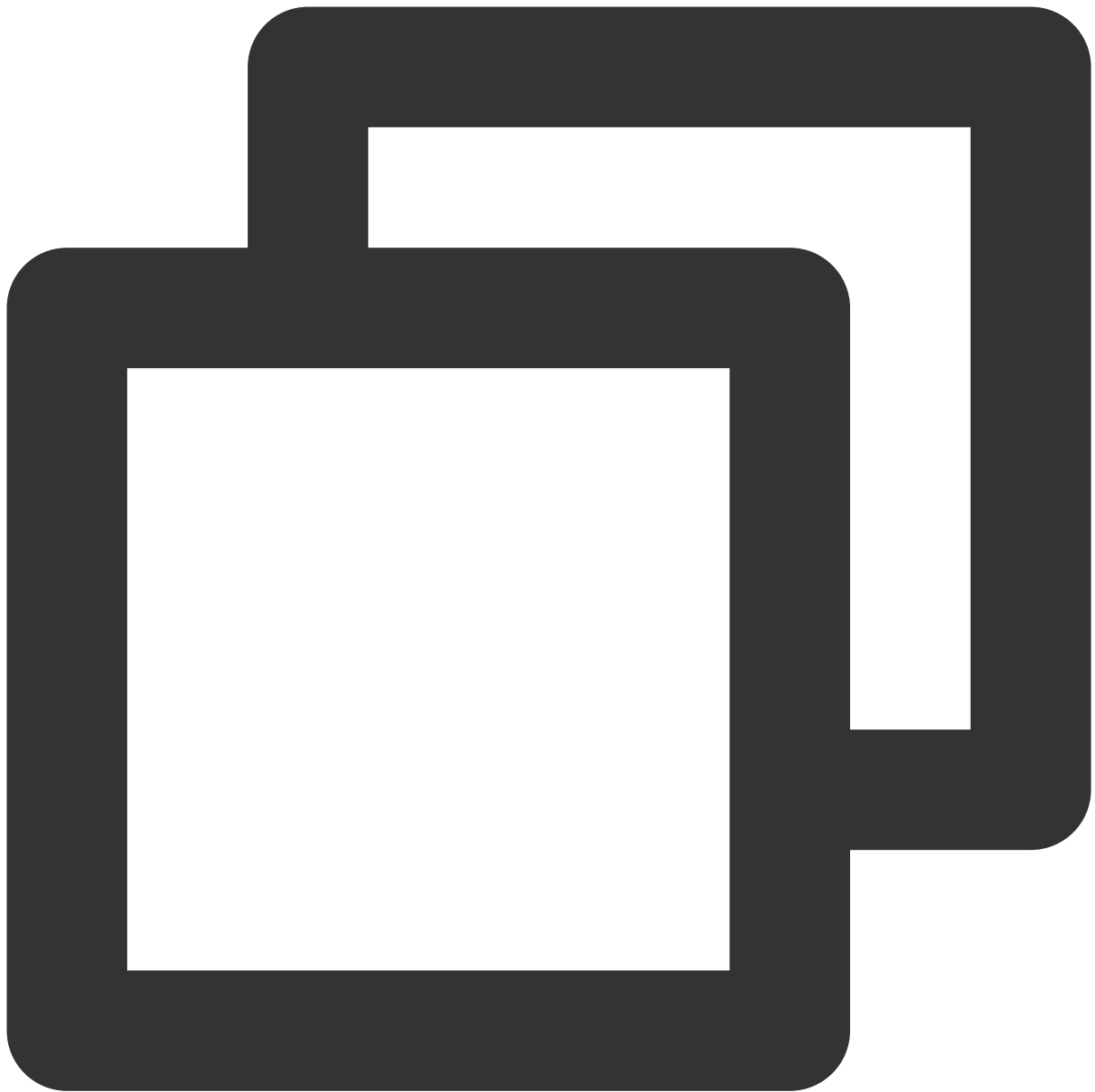


小程序 SDK 提供返回按钮自定义的功能，可以通过**自定义返回按钮**，**重写 AbsMiniUiProxy 的 navBarBackRes 方法实现**。

API 定义如下：

方法参数**mode**：表示导航栏标题颜色，1 表示黑色标题，0表示白色标题；

方法返回值：表示返回按钮的icon资源ID。

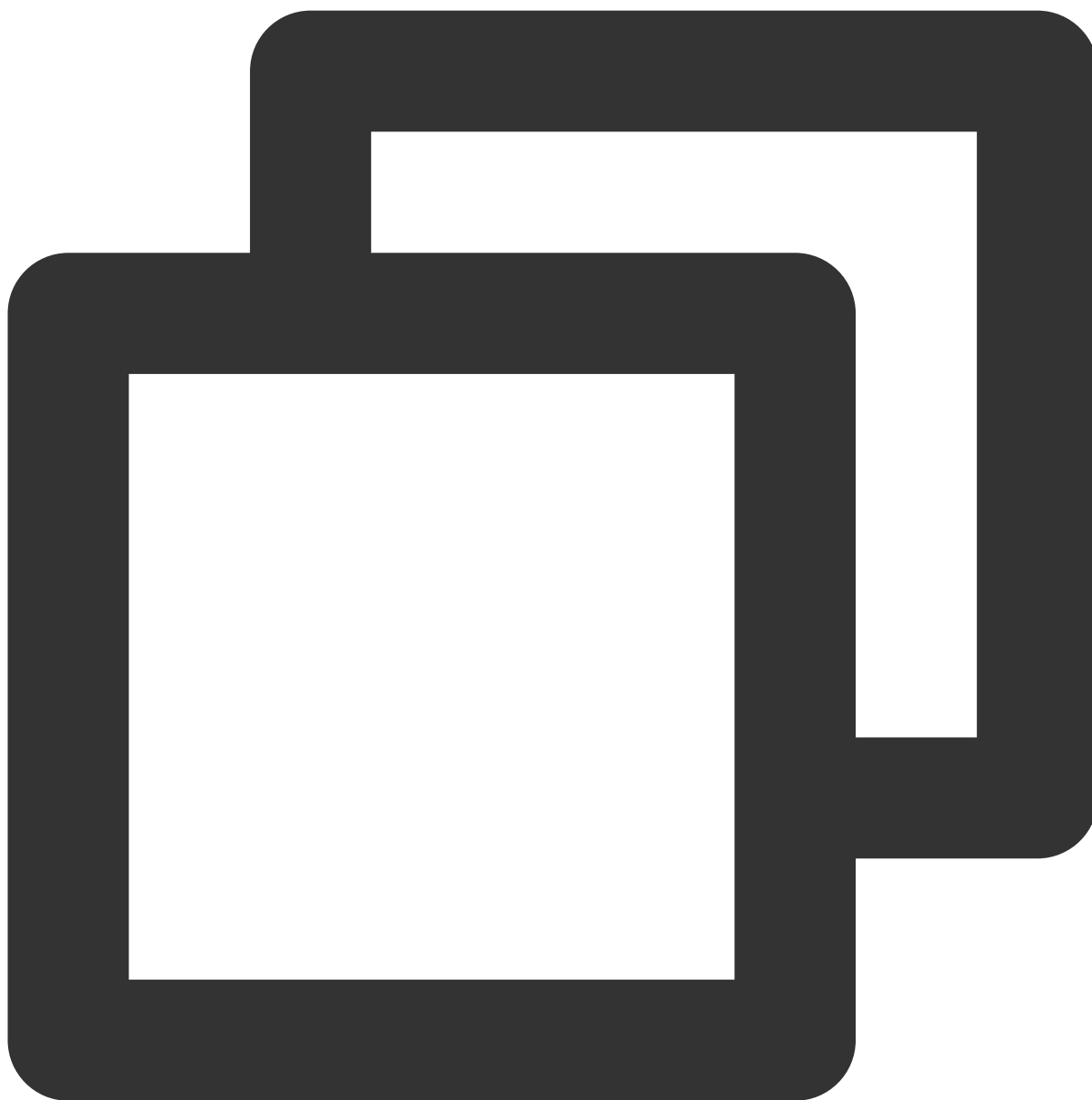


```
/**
 * 自定义导航栏返回icon，宽高比要求=24x43
 * 调用环境：子进程
 *
 * @param mode 导航栏标题颜色，1:black 0:white
 * @return
 */
@DrawableRes
int navBarBackRes(int mode);
```

注意：

返回按钮 icon 资源的宽高比要求为：24x43。

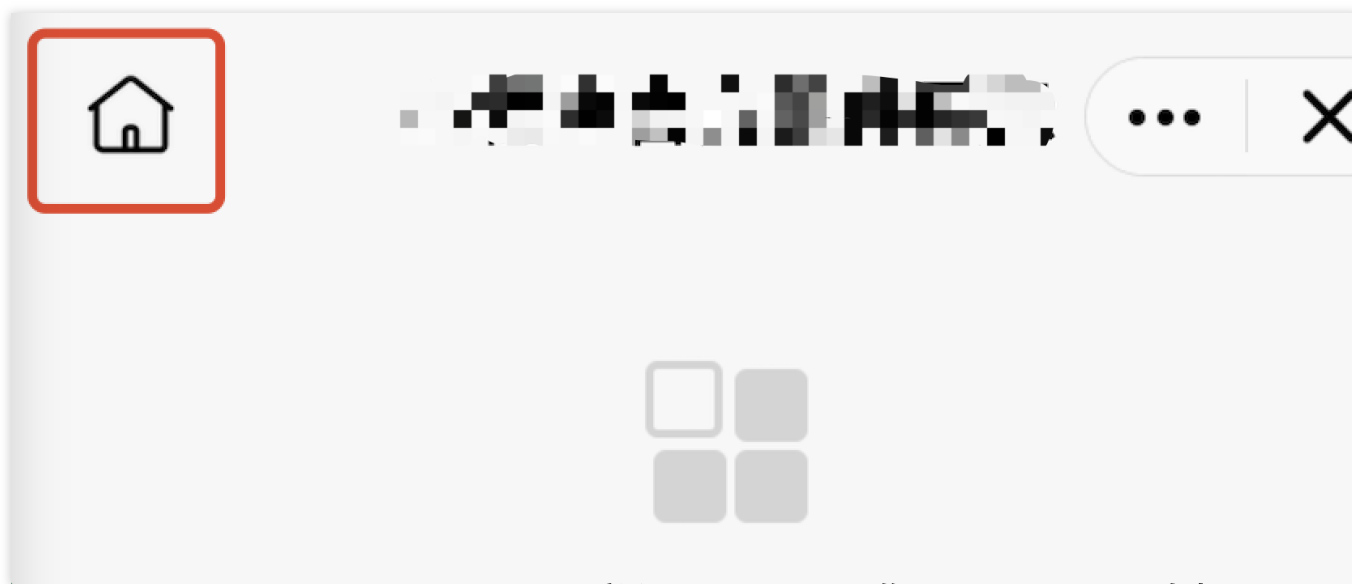
示例代码：



```
@Override
public int navBarBackRes(int mode) {
    if(mode == 0) { //black
        return R.drawable.back_icon; //your black icon res
    } else { //white
        return R.drawable.white_icon; //your white icon res
    }
}
```

2. home按钮

home 按钮示意图如下：

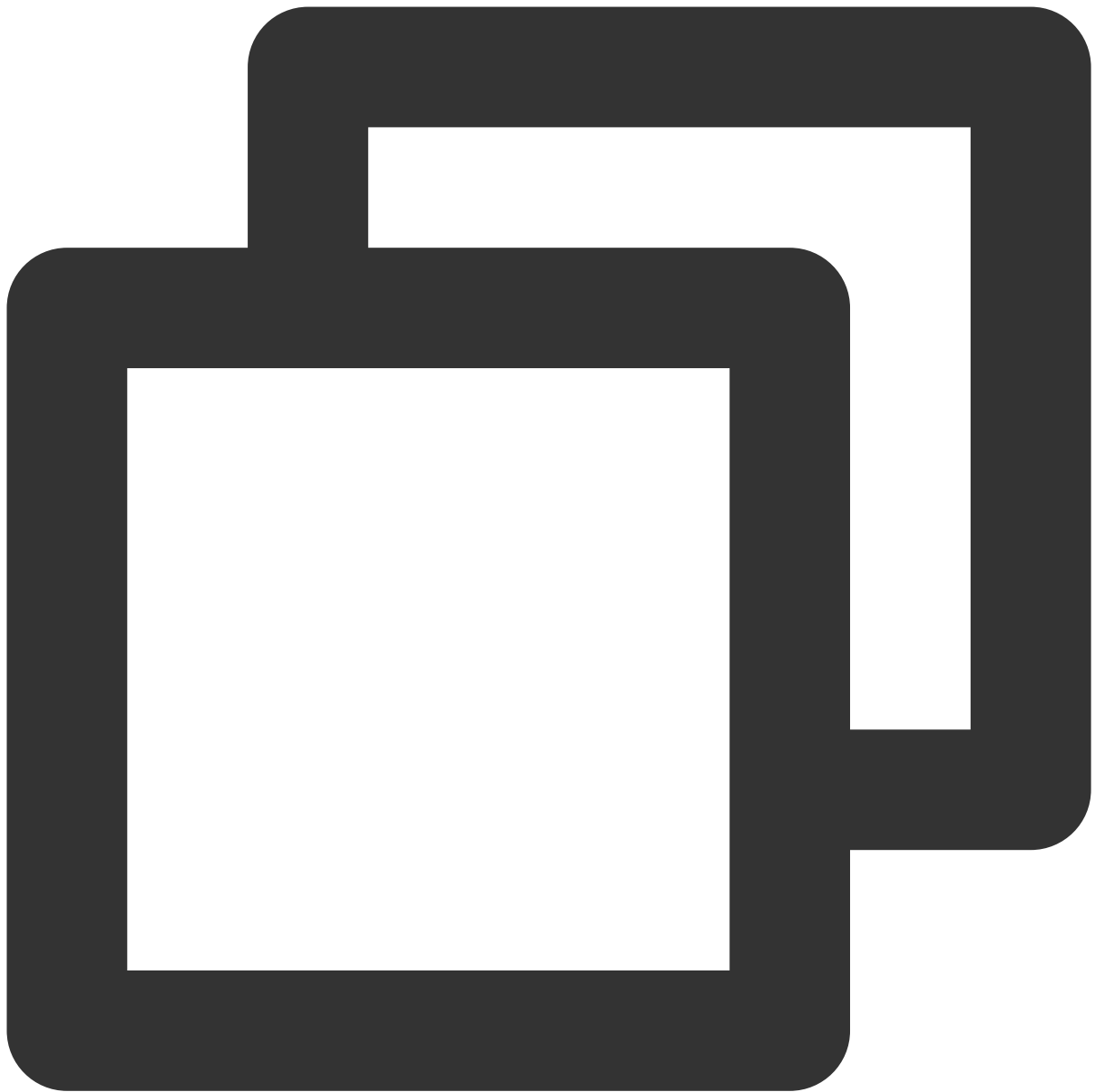


小程序 SDK 提供 home 按钮自定义的功能，可以通过**重写 AbsMiniUiProxy 的 homeButtonRes**，自定义 home 按钮方法实现。

API 定义如下：

方法参数 **mode**：表示导航栏标题颜色，1 表示黑色标题，0 表示白色标题；

方法返回值：表示返回按钮的 icon 资源 ID。

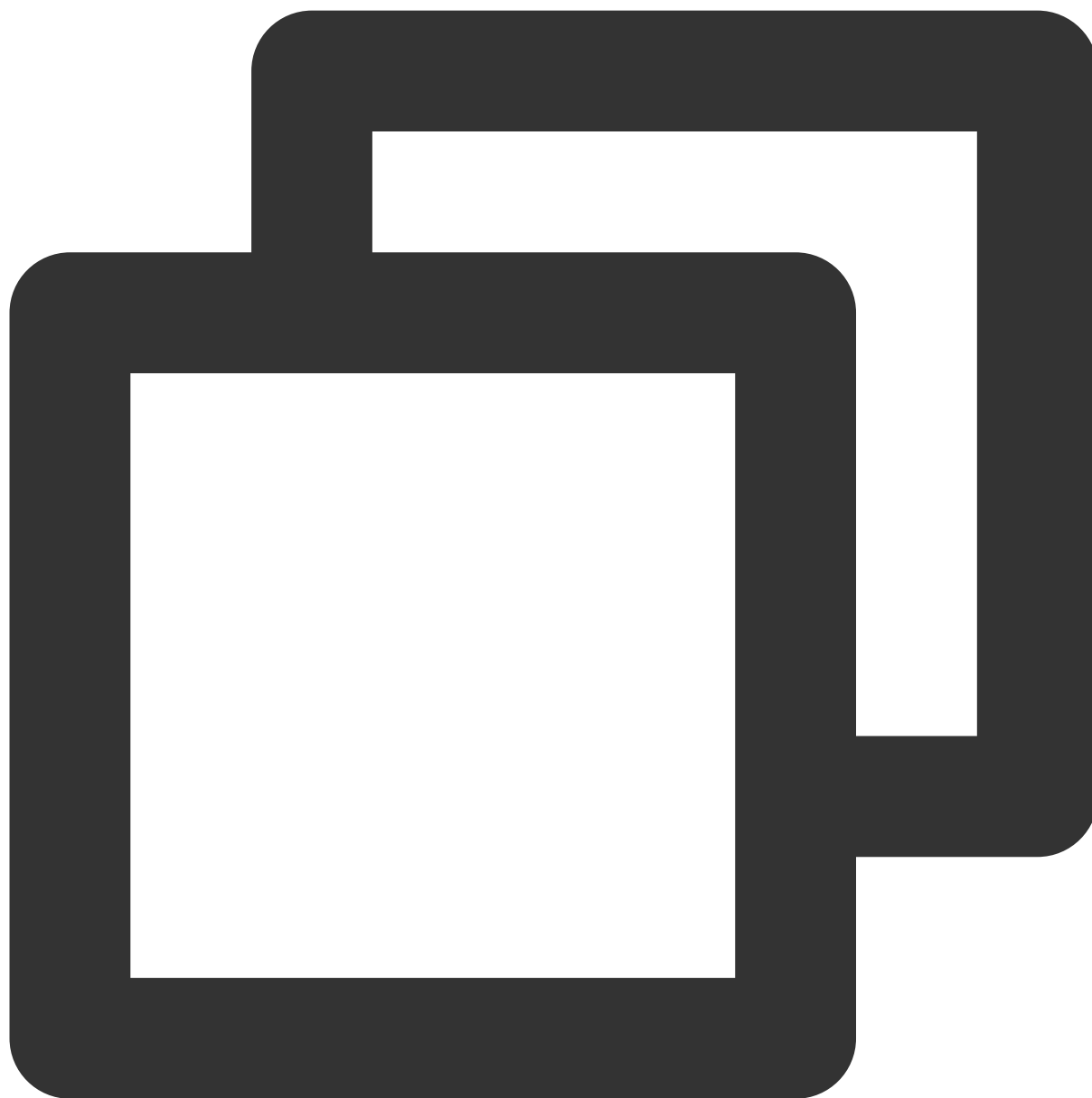


```
/**
 * 导航栏返回主页图标icon，宽高比要求=48x48
 * 调用环境：子进程
 *
 * @param mode 导航栏标题颜色，1:black 0:white
 * @return
 */
@DrawableRes
int homeButtonRes(int mode);
```

注意：

尺寸要求：导航栏 home 图标 icon, 宽高比要求=48x48。

示例代码：



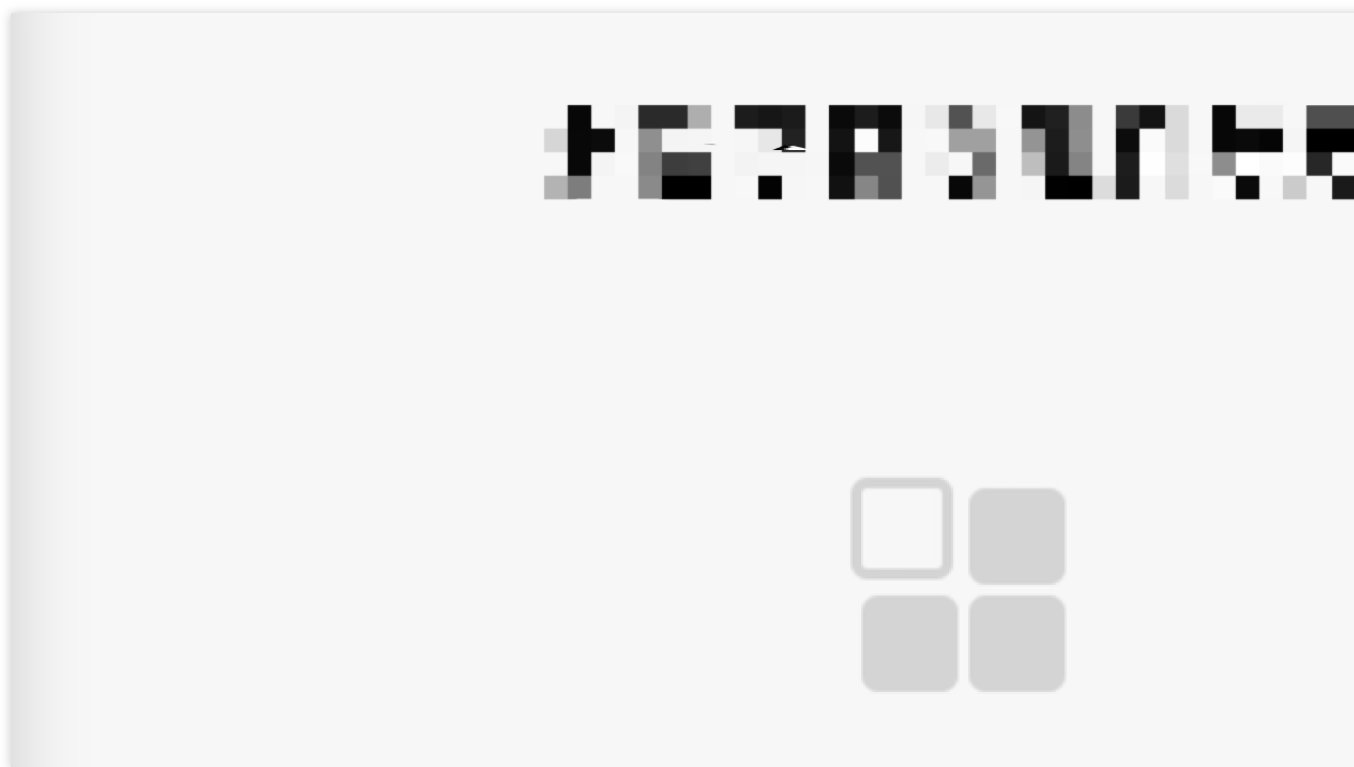
```
@Override
public int homeButtonRes(int mode) {

    if(mode == 0) { //black
        return R.drawable.home_black_icon;
    } else {
        return R.drawable.home_white_icon;
    }
}
```

```
}  
}
```

3. 更多按钮

更多按钮展示示意图如下：

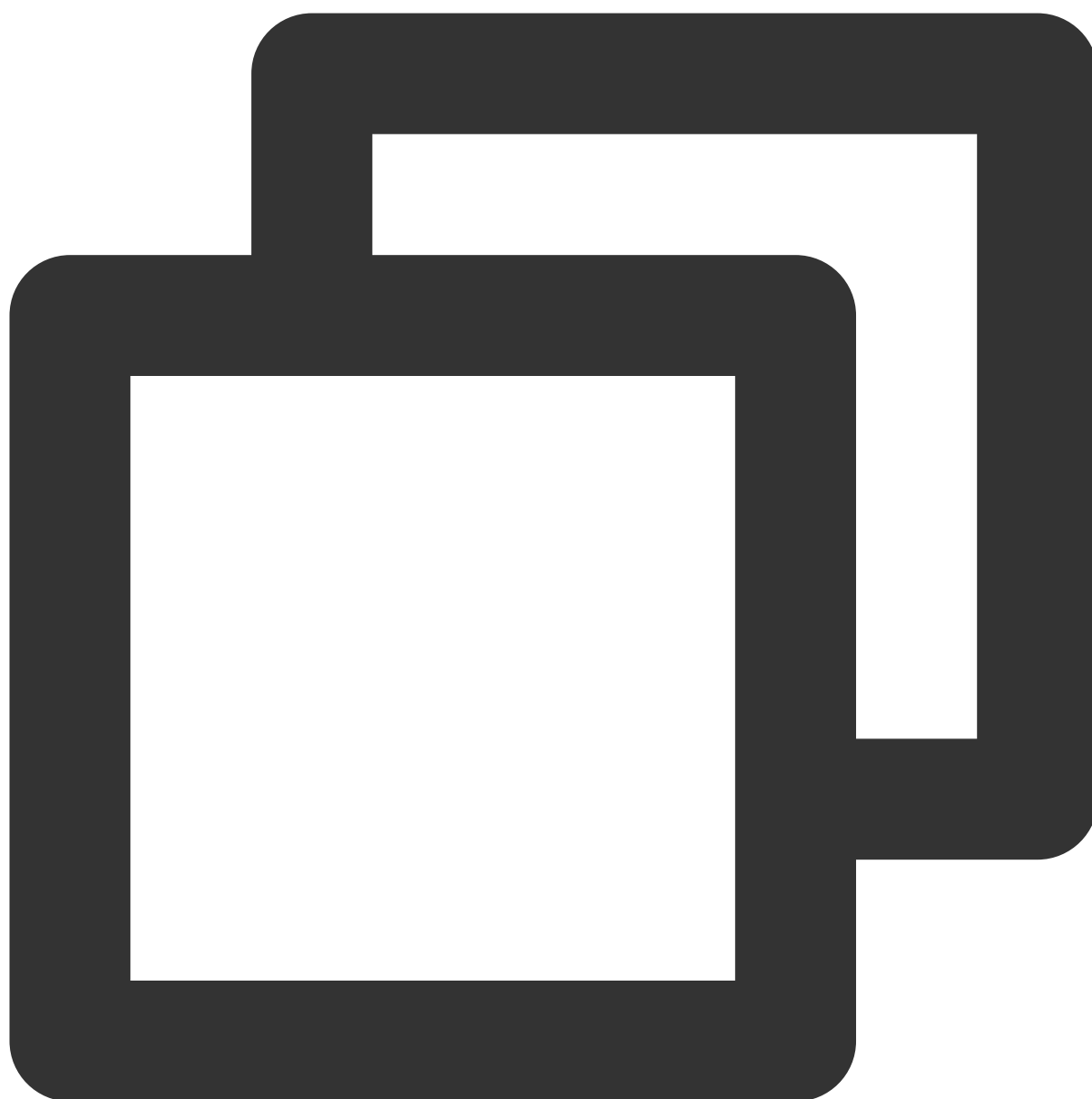


小程序SDK提供更多按钮自定义的功能，可以通过**重写 AbsMiniUiProxy 的 moreButtonRes**，自定义 **home 按钮方法实现**。

API定义如下：

方法参数**mode**：表示导航栏标题颜色，1 表示黑色标题，0表示白色标题；

方法返回值：表示返回按钮的**icon**资源ID。

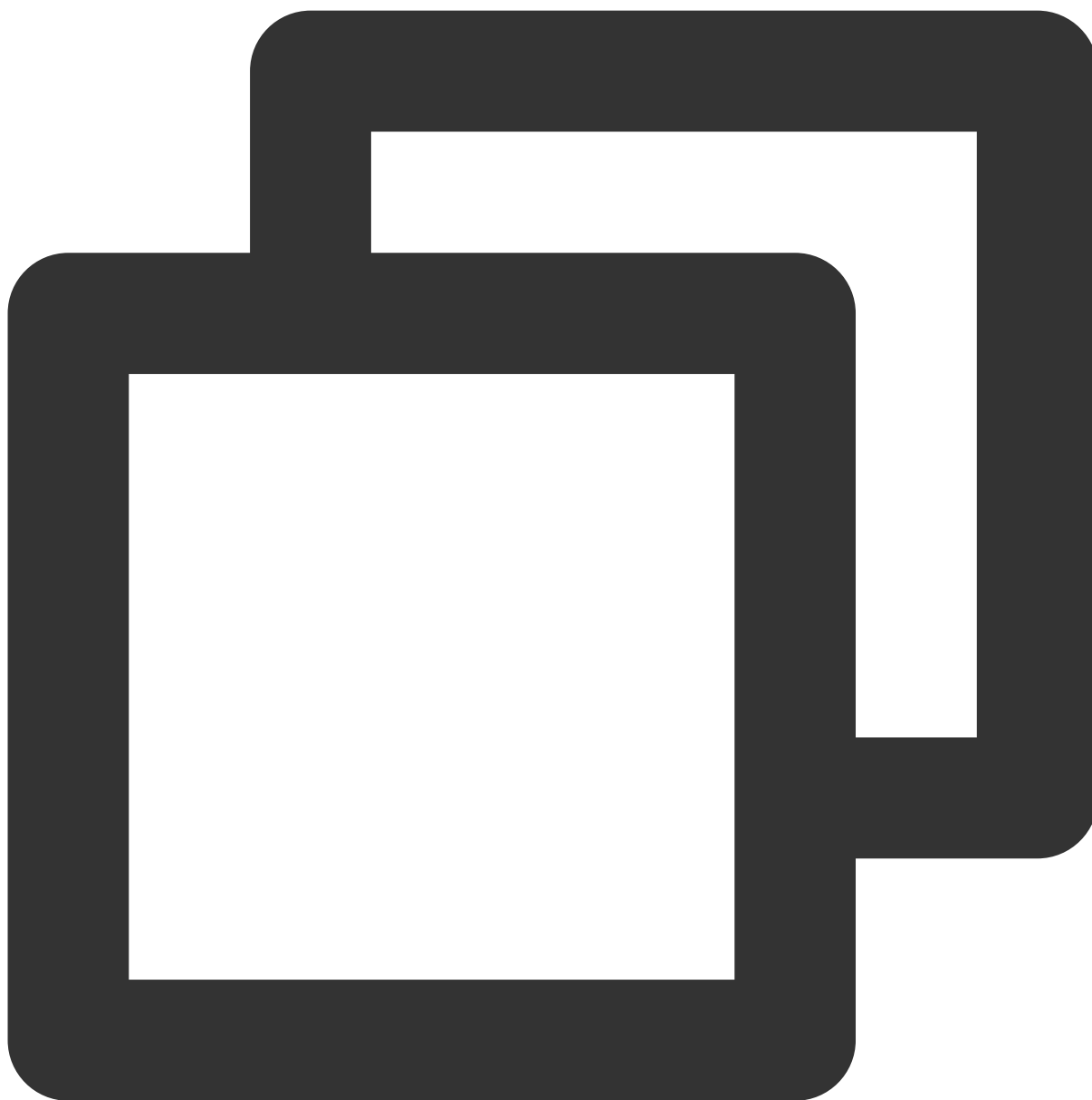


```
/**
 * 导航栏返回主页图标icon，宽高比要求=48x48
 * 调用环境：子进程
 *
 * @param mode 导航栏标题颜色，1:black 0:white
 * @return
 */
@DrawableRes
int moreButtonRes(int mode);
```

注意：

尺寸要求：导航栏 home 图标 icon, 宽高比要求=80x59。

示例代码：



```
@Override
public int moreButtonRes(int mode) {

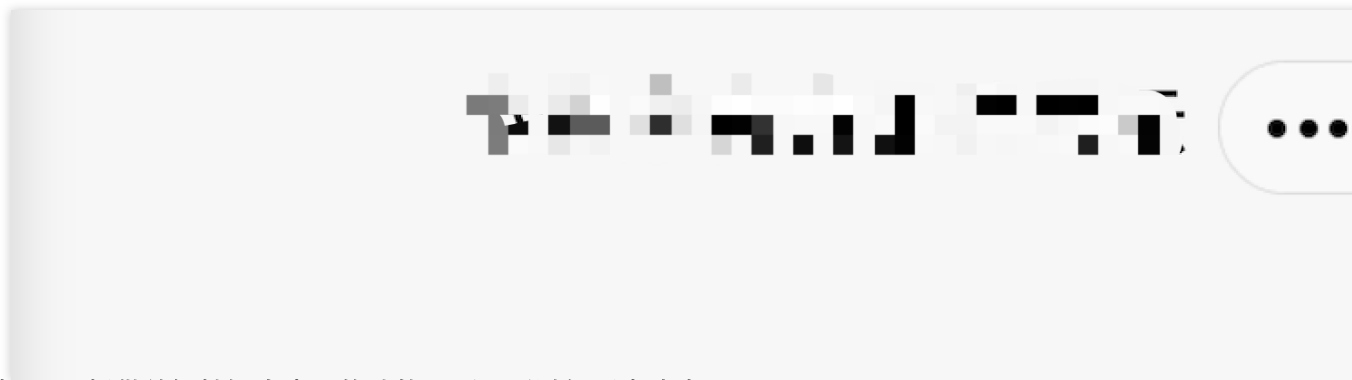
    if(mode == 0) { //black
        return R.drawable.more_black_icon;
    } else {
        return R.drawable.more_white_icon;
    }
}
```



```
}  
}
```

4. 小程序关闭按钮

小程序关闭按钮示意图如下：



小程序 SDK 提供关闭按钮自定义的功能，可以通过如下方式实现：

自定义 home 按钮：

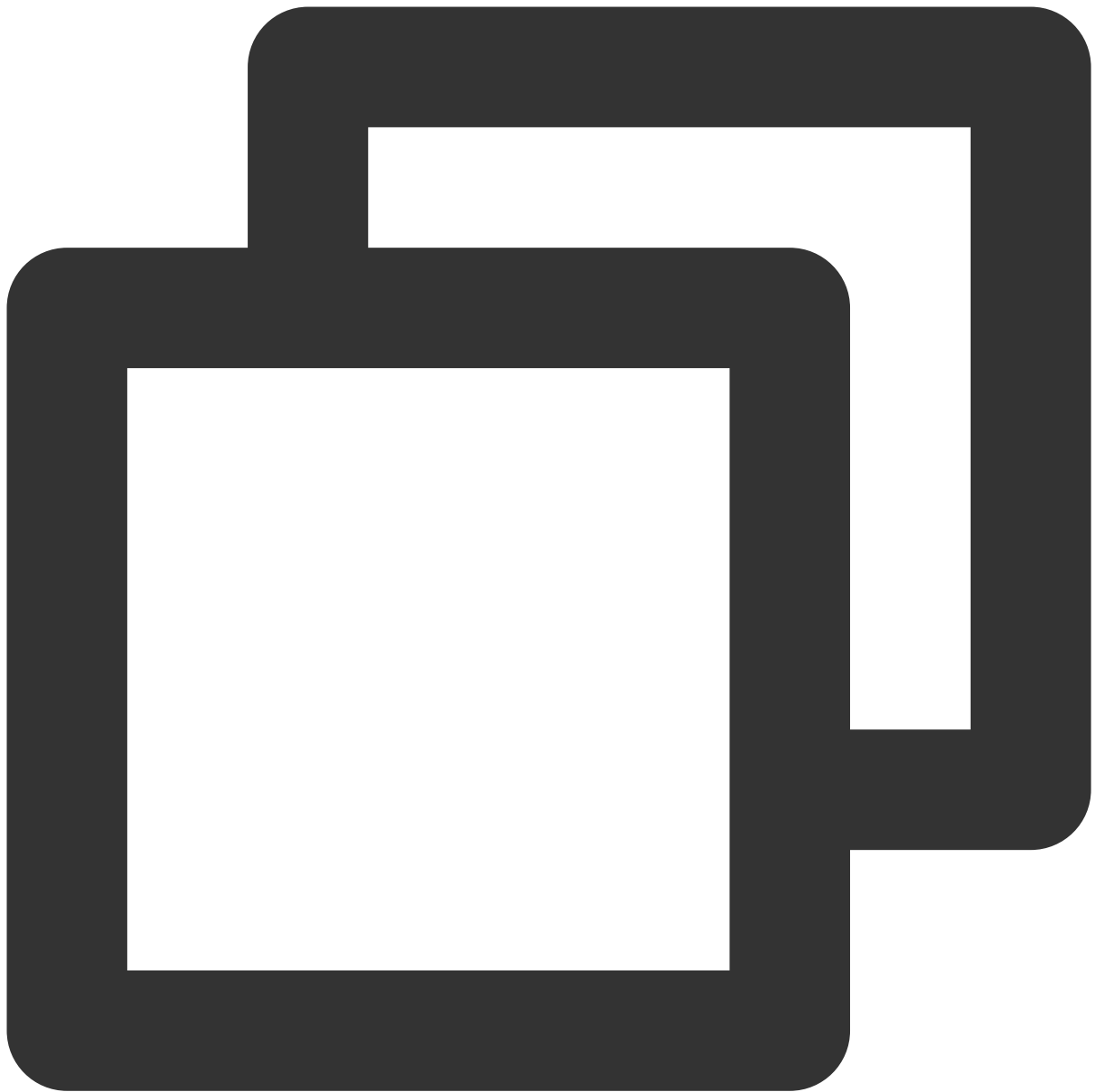
重写 `AbsMiniUiProxy` 的 `closeButtonRes` 方法。

API 定义如下：

API 说明：

方法参数 `mode`：表示导航栏标题颜色，1 表示黑色标题，0 表示白色标题。

方法返回值：表示返回按钮的 icon 资源 ID。

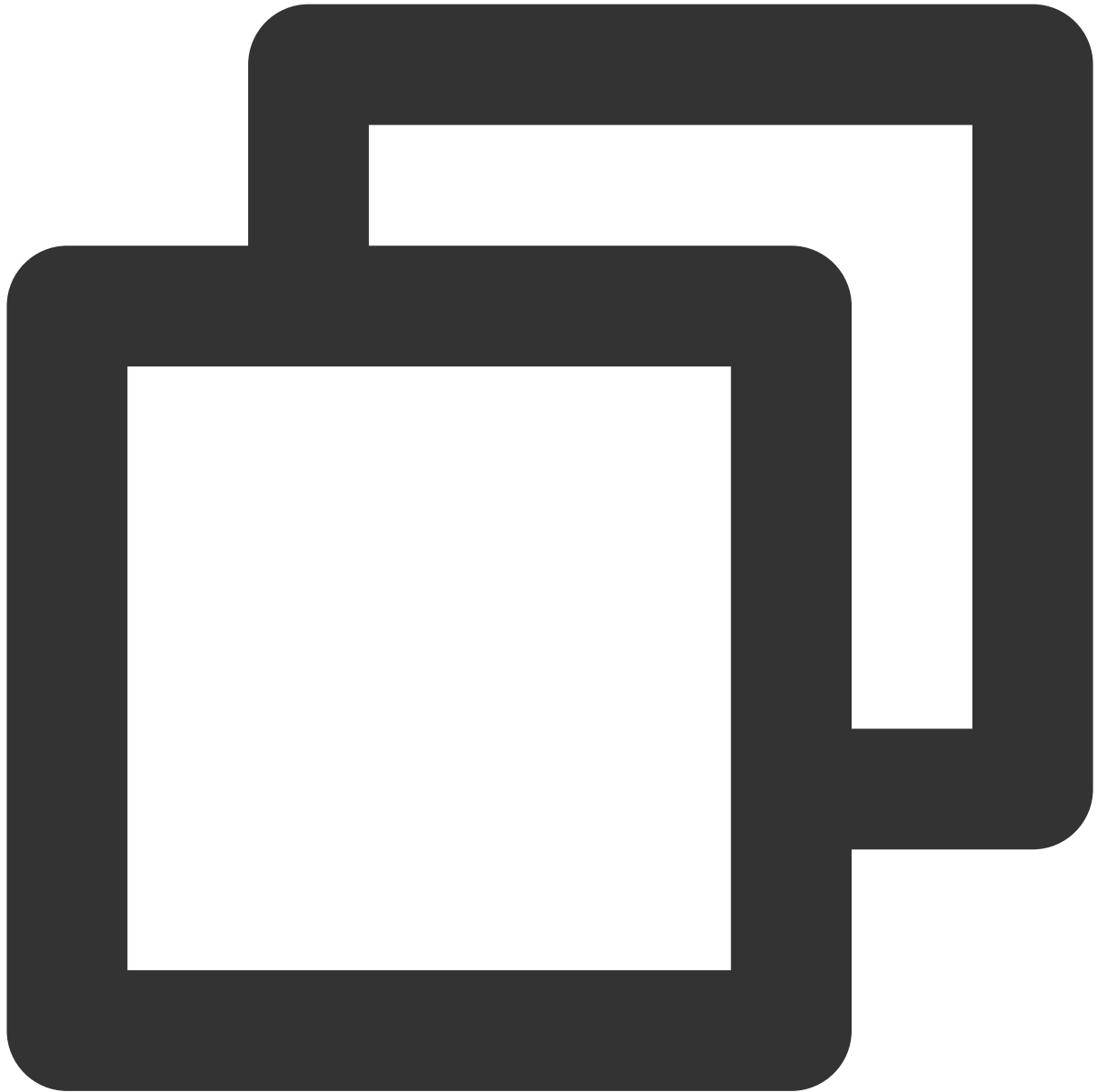


```
/**
 * 导航栏返回主页图标icon，宽高比要求=48x48
 * 调用环境：子进程
 *
 * @param mode 导航栏标题颜色，1:black 0:white
 * @return
 */
@DrawableRes
int closeButtonRes(int mode);
```

注意：

尺寸要求：导航栏home图标icon, 宽高比要求=80x59。

示例代码：



```
@Override
public int closeButtonRes(int mode) {

    if(mode == 0) {//black
        return R.drawable.close_black_icon;
    }else {
        return R.drawable.close_white_icon;
    }
}
```

```
}  
}
```

5. 胶囊按钮分割线

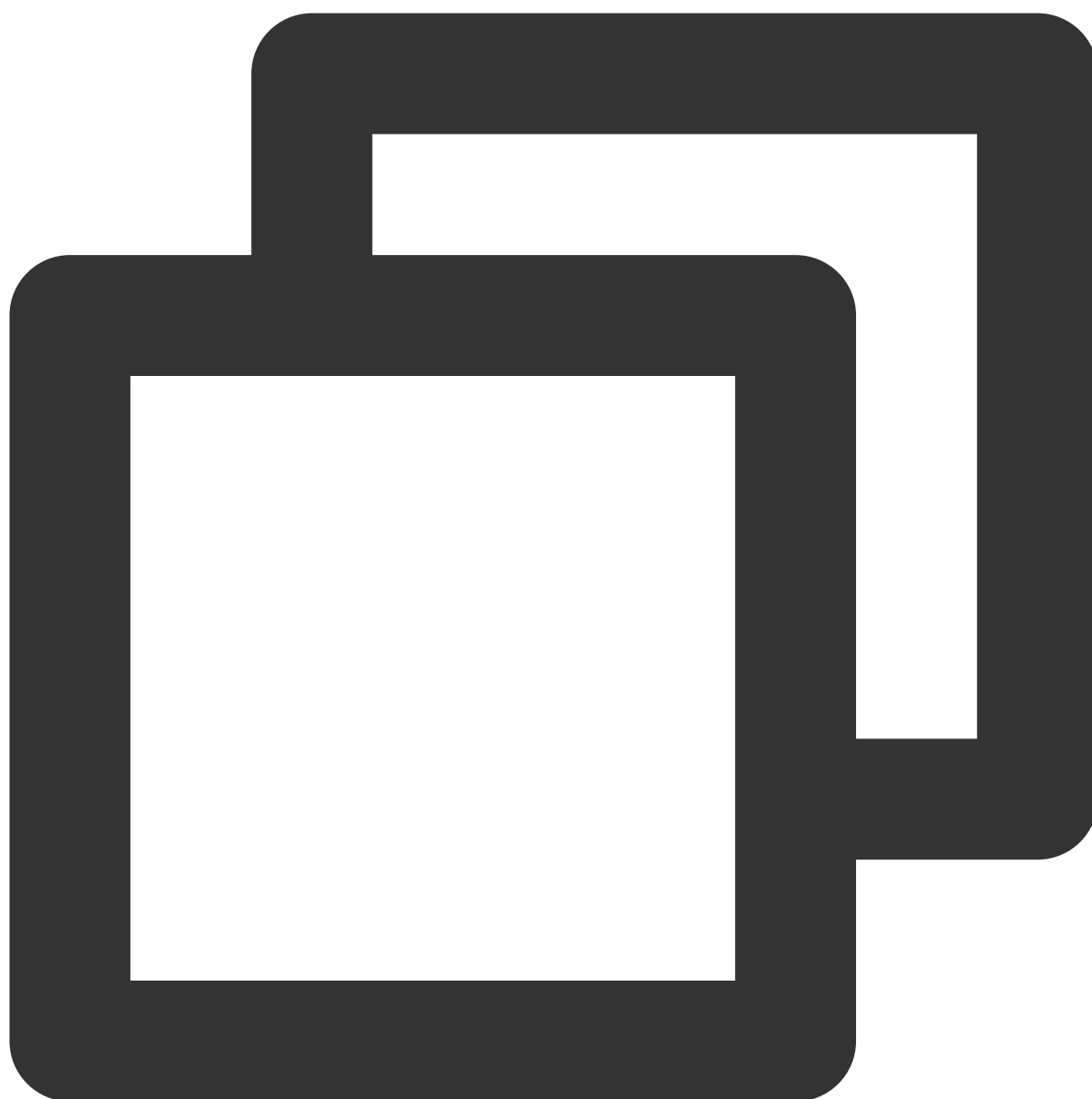
小程序胶囊按钮分割线展示意图如下：



小程序 SDK 提供自定义胶囊按钮分割线的功能，可以通过**重写 AbsMiniUiProxy 的 lineSplitBackgroundColor**，自定义 **home 按钮方法实现**。

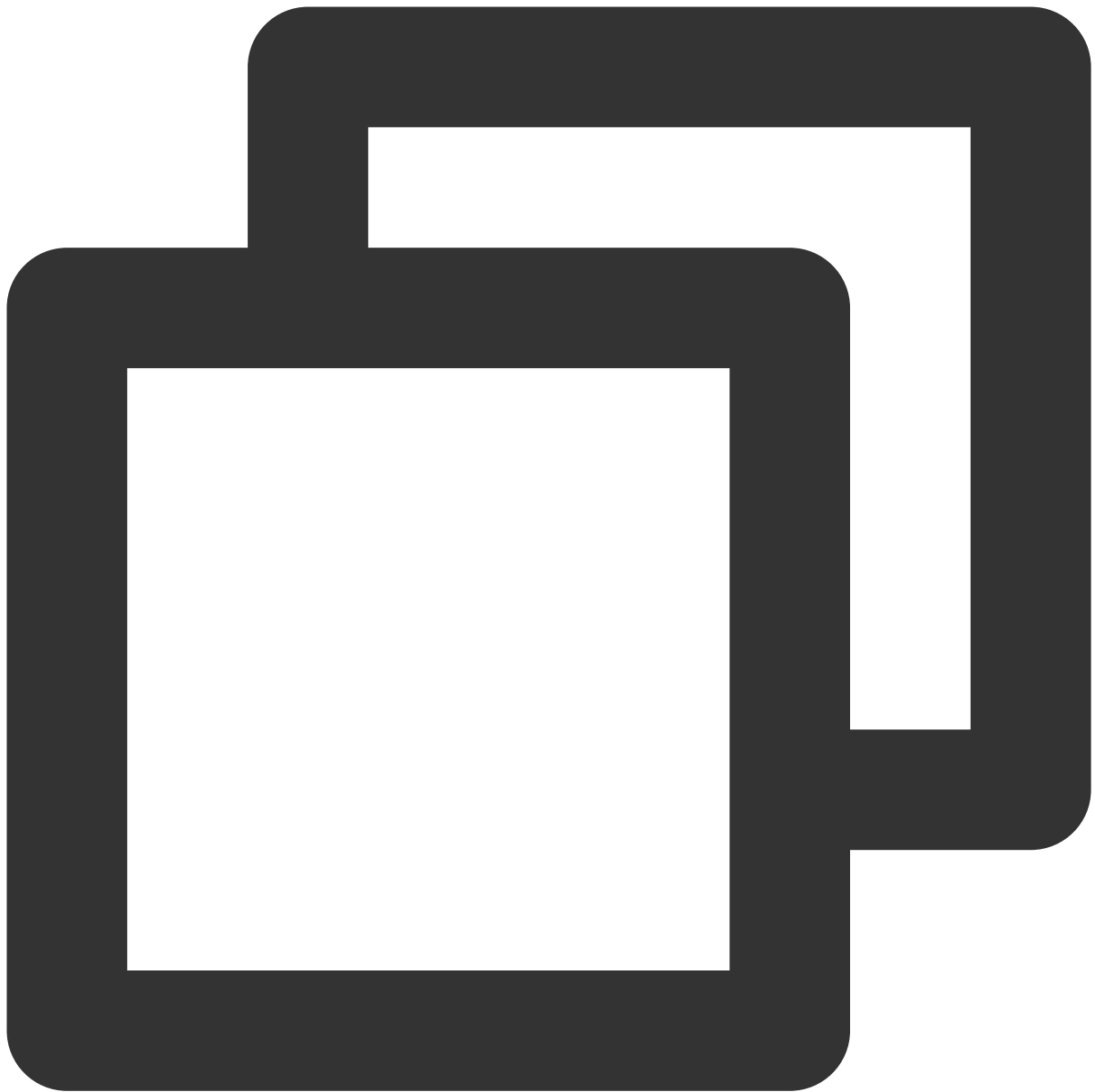
API 定义如下：

方法返回值：表示分割线背景颜色。



```
/**
 * 胶囊按钮中间分割线背景颜色
 * 调用环境：子进程
 *
 * @return
 */
@DrawableRes
int lineSplitBackgroundColor();
```

示例代码：



```
@Override
public int lineSplitBackgroundColor() {
    return Color.RED;
}
```

定制授权弹窗

当小程序调用的 API 需要授权时，SDK 提供如下默认的授权 UI 样式：



Get your location

The applet obtains location information and will provide you with positioning and other services

Reject

Allow

开发者也可以通过如下方法自定义授权 UI 样式。

自定义授权弹窗：

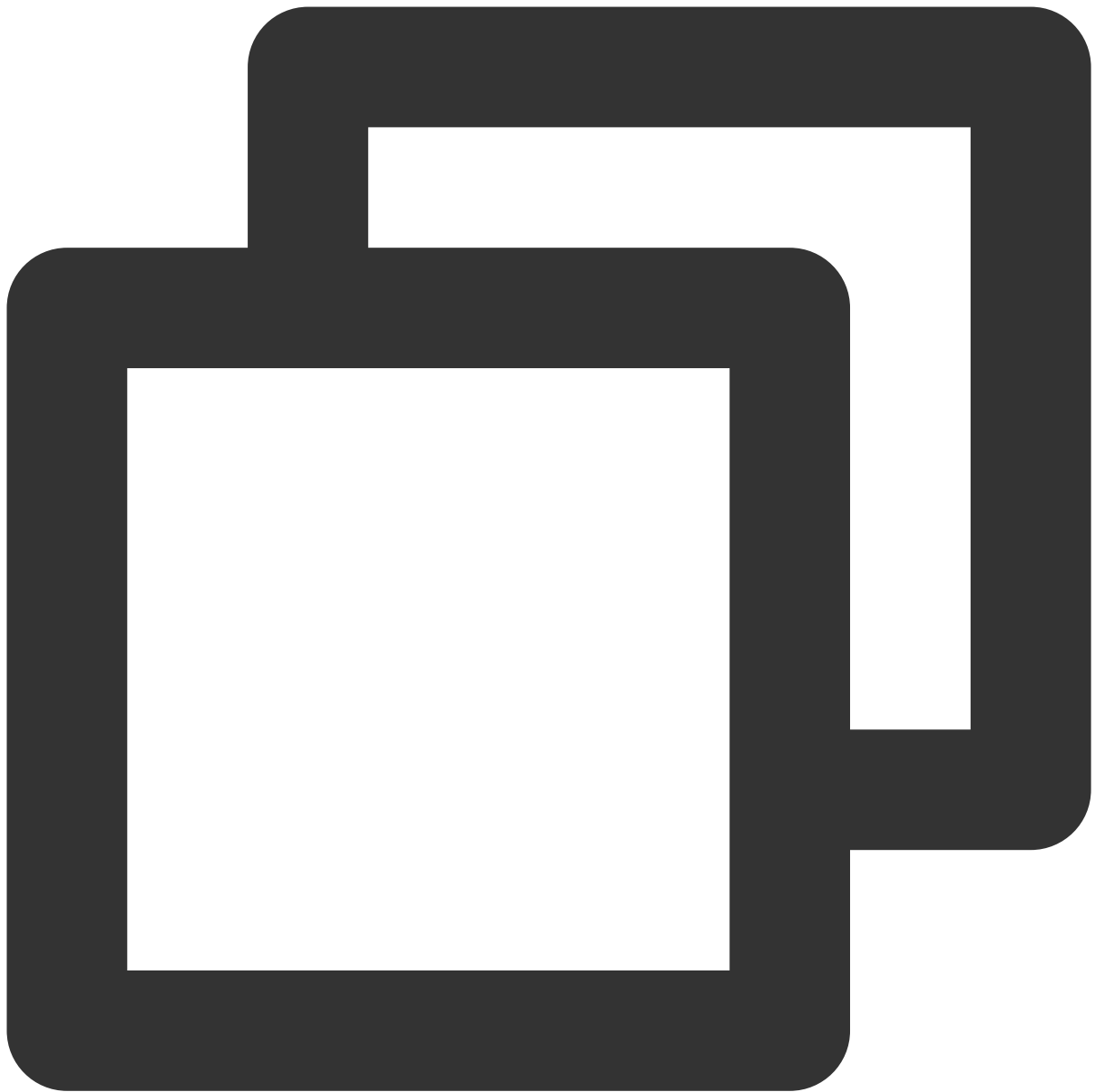
重写 `AbsMiniUiProxy` 的 `authView` 方法；

参数 `Context` 表示小程序进程的上下文；

参数 `MiniAuthInfo` 包含有当前小程序所请求的授权信息，详见 `MiniAuthInfo`；

参数 `IAuthView`，用于设置自定义的弹窗 `View`。

API 定义如下：



```
/**
 * 自定义授权弹窗view
 * 调用环境：子进程
 *
 * @param context
 * @param authInfo
 * @param authView
 * @return true:自定义授权view;false:使用内置
 */
@Override
public boolean authView(Context context, MiniAuthInfo authInfo, IAuthView authView)
```

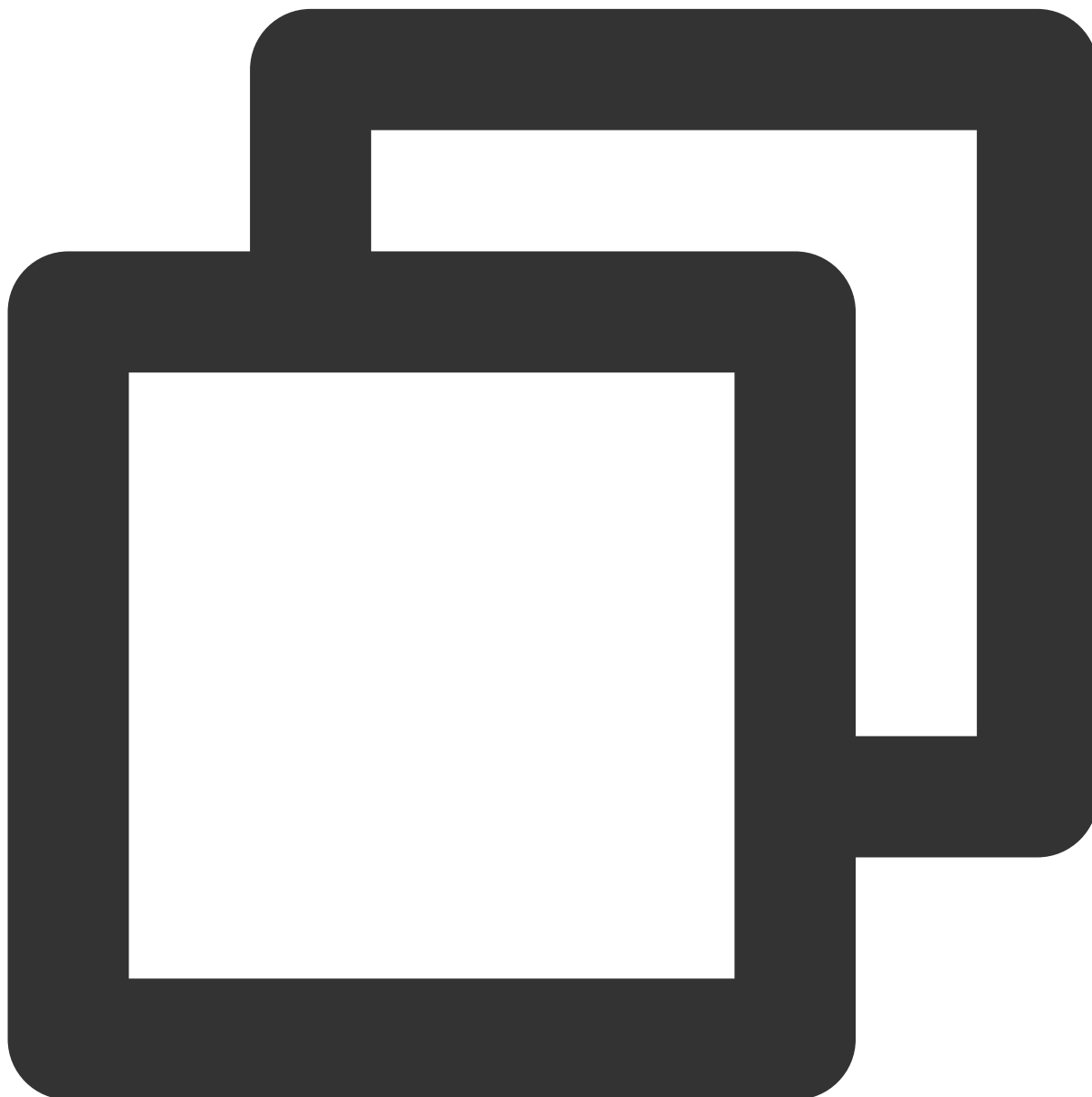

注意：

`authView` 方法的返回值表示是否使用自定义授权视图，返回值为 `false` 使用默认内置的授权视图，返回值为 `true` 使用自定义授权视图。

`MiniAuthInfo` 中的 `refuseListener` 和 `grantListener` 表示授权弹窗的监听，必须设置，否则会导致小程序授权异常。

示例代码如下

视图布局文件



```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
android:layout_width="match_parent"
android:layout_height="match_parent"
android:paddingLeft="16dp"
android:paddingTop="16dp"
android:paddingRight="16dp"
android:paddingBottom="15dp"
android:background="@android:color/white">

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="custom auth view"
    android:layout_centerHorizontal="true"/>

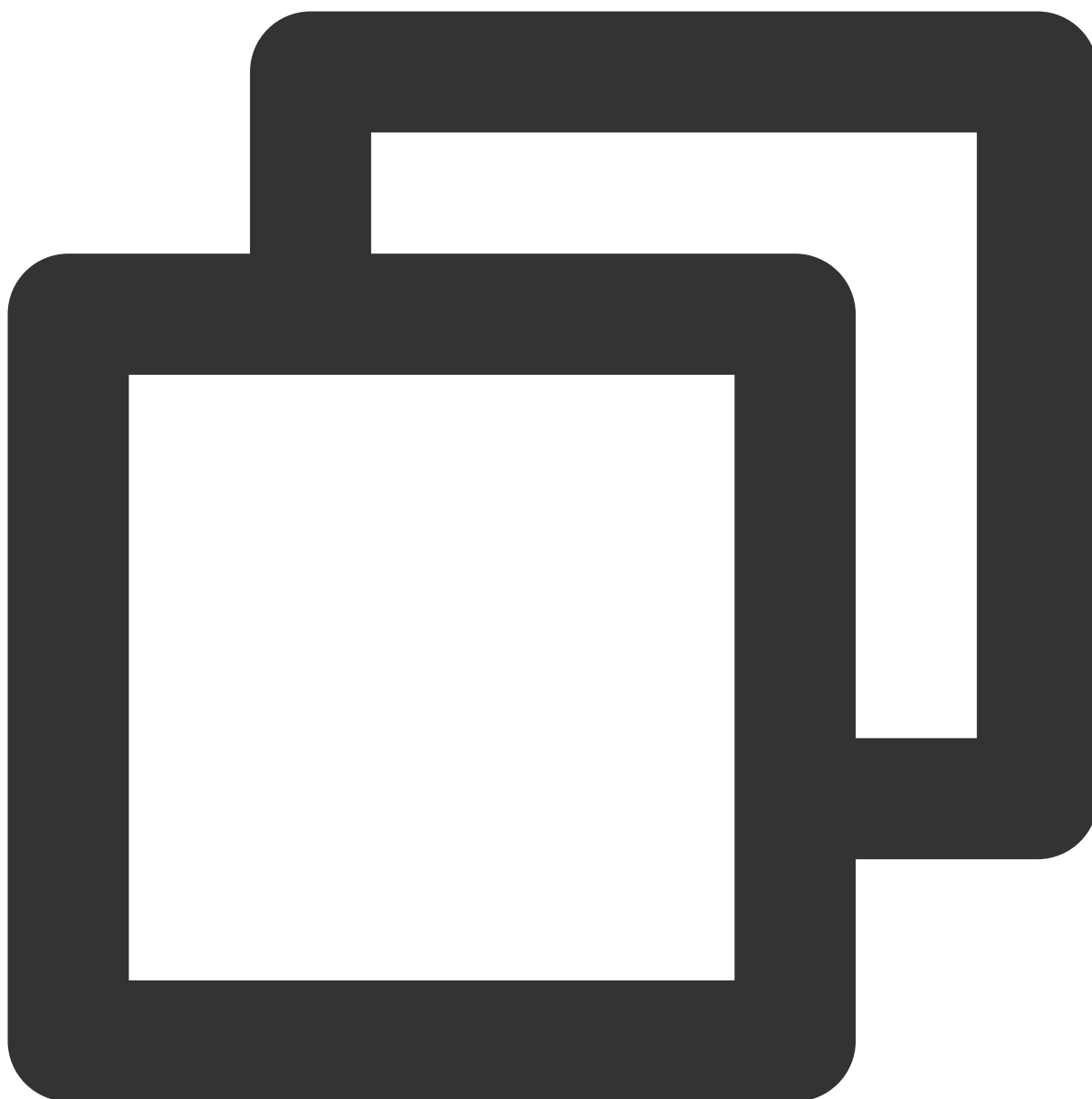
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="150dp">

    <Button
        android:id="@+id/mini_auth_btn_refuse"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1.0"
        android:text="@string/applet_mini_reject" />

    <Button
        android:id="@+id/mini_auth_btn_grant"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1.0"
        android:text="@string/applet_mini_auth" />
</LinearLayout>

</RelativeLayout>
```

重写 `authView` 方法（注意方法返回值为 `true`）：

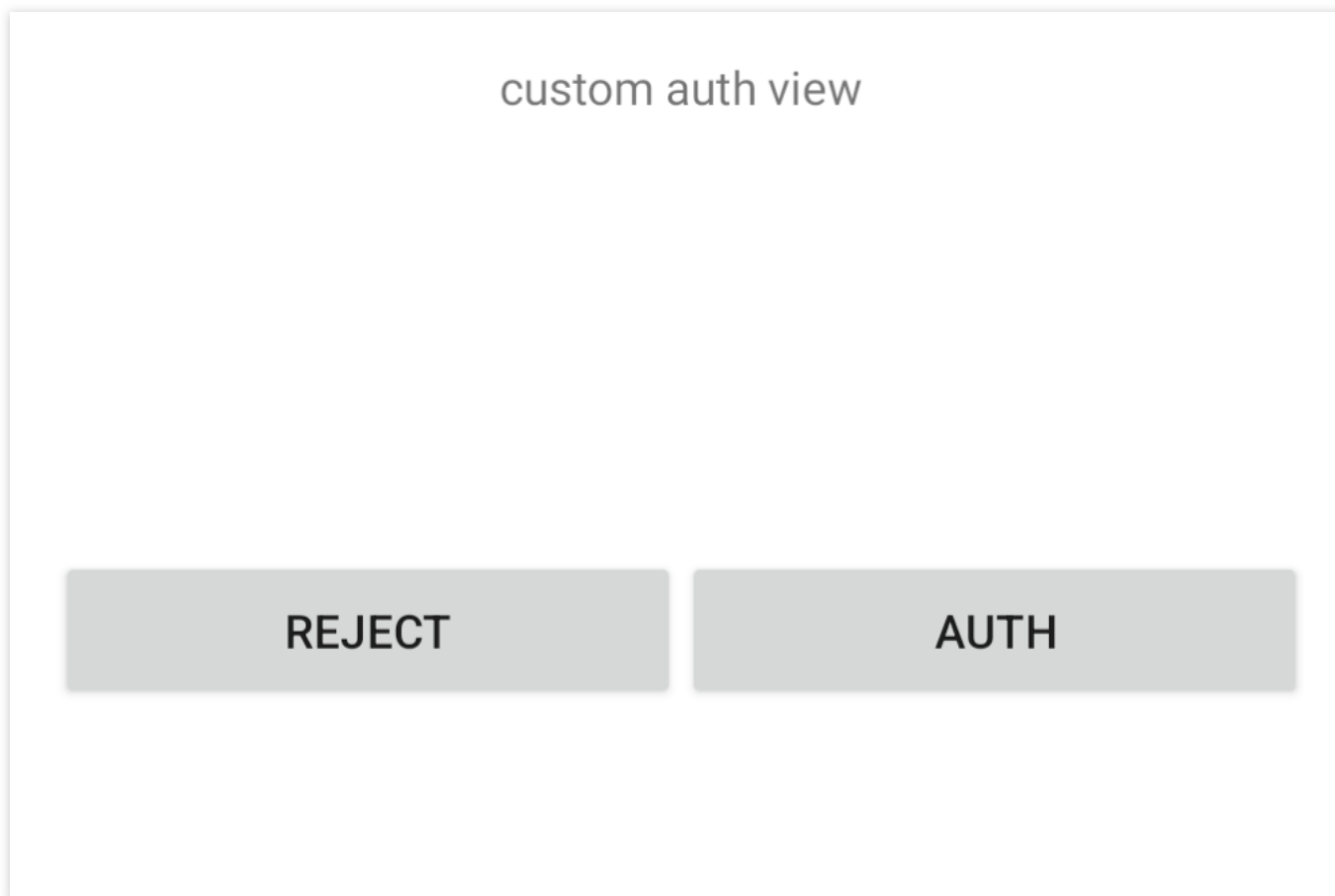


```
@Override
public boolean authView(Context context, MiniAuthInfo authInfo, IAuthView authView)
    boolean isCustom = true;
    if (isCustom) {
        View view = LayoutInflater.from(context).inflate(R.layout.mini_auth_view, n
        //必须设置
        view.findViewById(R.id.mini_auth_btn_refuse).setOnClickListener(authInfo.re
        //必须设置
        view.findViewById(R.id.mini_auth_btn_grant).setOnClickListener(authInfo.gra

        //返回自定义View
```

```
    authView.getView(view);  
  }  
  
  return isCustom;  
}
```

自定义效果：

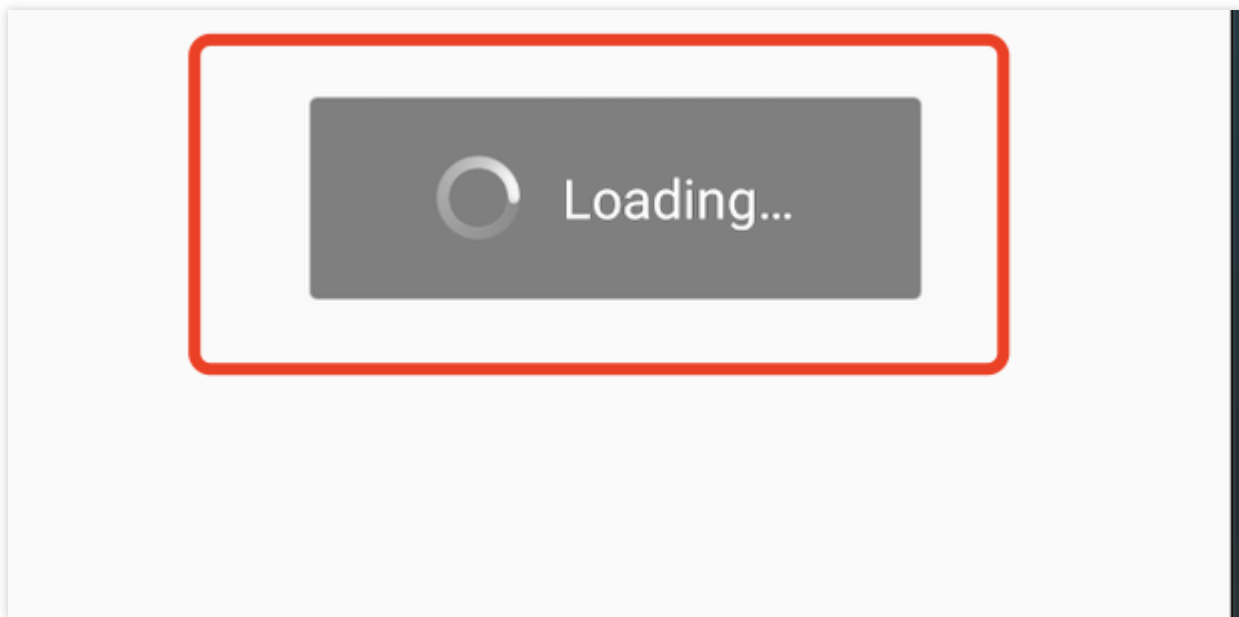


定制小程序 Loading 视图

小程序打开过程中有检查更新和启动加载 Loading 动画，这两种 Loading 动画都支持自定义。

1. 定制小程序检查更新 Loading 视图

默认检查更新的 Loading 动画如下图所示：



可以通过重写 **AbsMiniUiProxy** 的 **updateLoadingView** 方法检查更新动画的自定义。

API 描述如下：

说明：

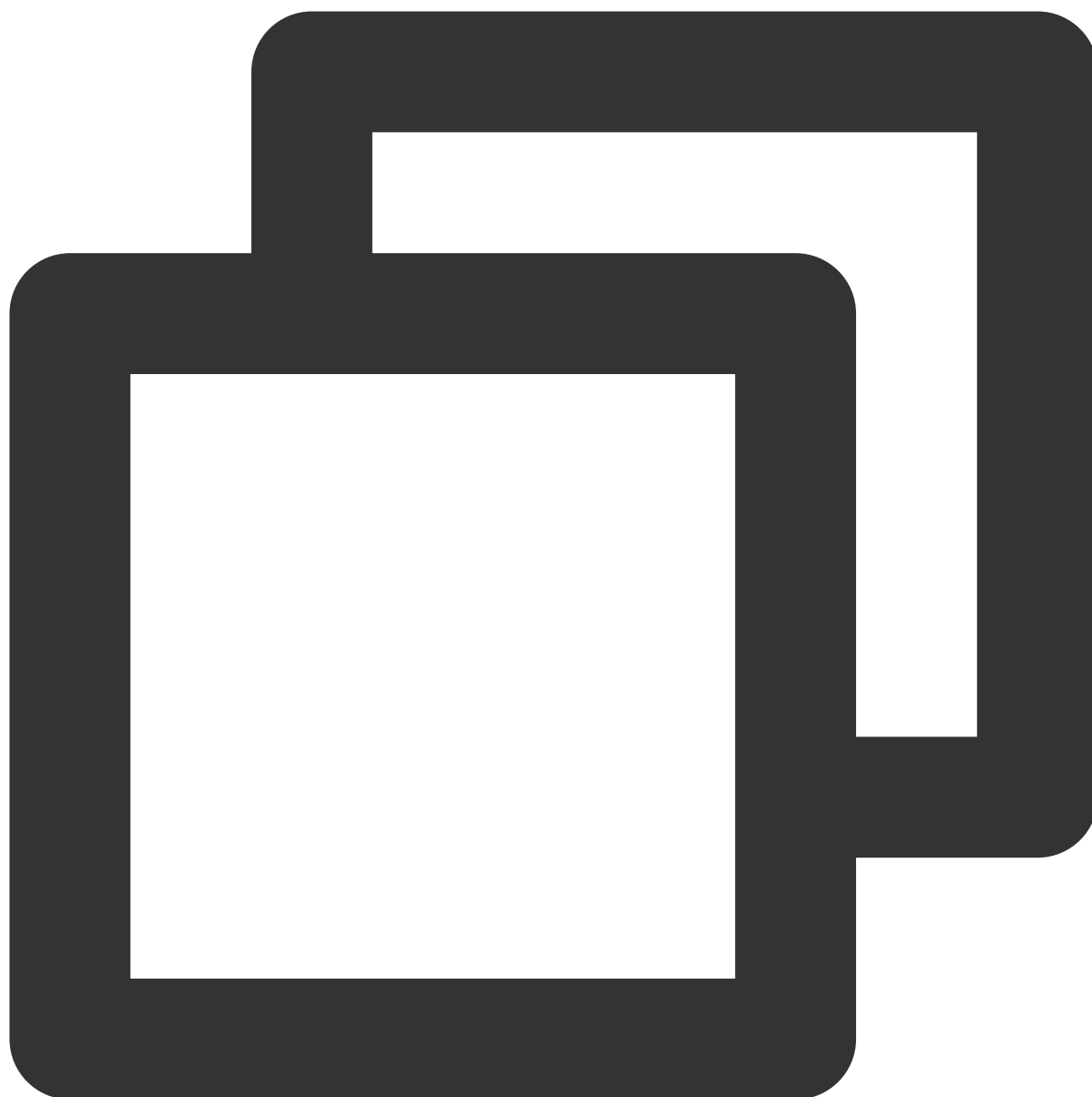
updateLoadingView 方法的返回值是 **IMiniLoading** 类型的实例。

IMiniLoading 的方法说明：

create 创建一个 Loading 视图；

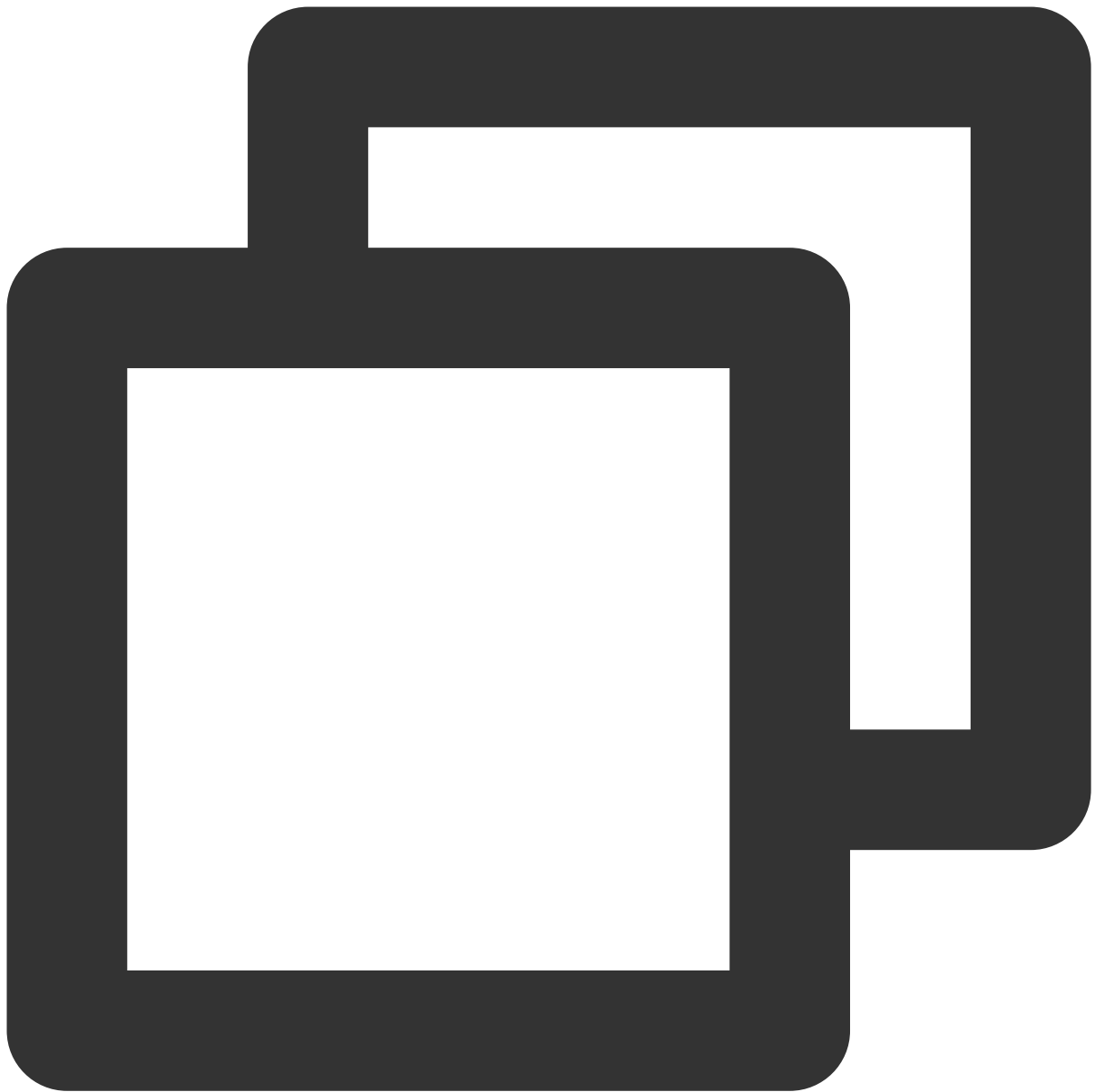
show 展示 Loading 效果时回调；

stop 停止 Loading 效果时回调。



```
/**
 * 自定义小程序检查更新loading页面
 * 调用环境：主进程
 *
 * @param context
 * @return
 */
public abstract IMiniLoading updateLoadingView(Context context);
```

示例代码：



```
@Override
public IMiniLoading updateLoadingView(Context context) {
    return new IMiniLoading() {
        @Override
        public View create() {
            return LayoutInflater.from(context).inflate(R.layout.applet_activity_cu
        }

        @Override
        public void show(View v) {
```

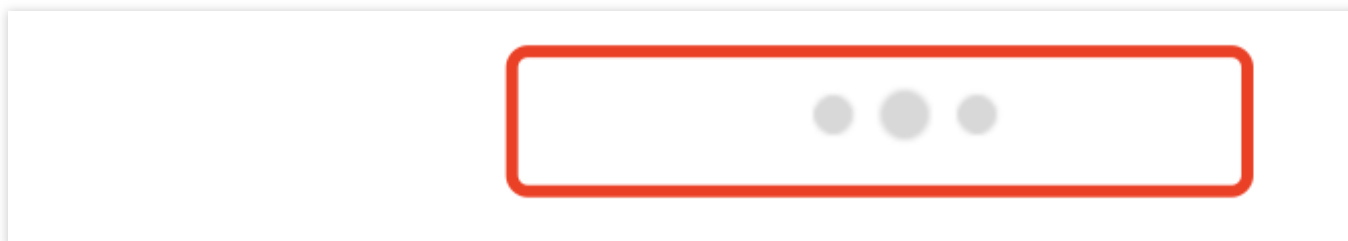
```
    }

    @Override
    public void stop(View v) {

    }

};
}
```

2. 定制小程序加载 Loading 视图



可以通过重写 **AbsMiniUiProxy** 的 **startLoadingView** 方法检查更新动画的自定义。

API 描述如下：

说明：

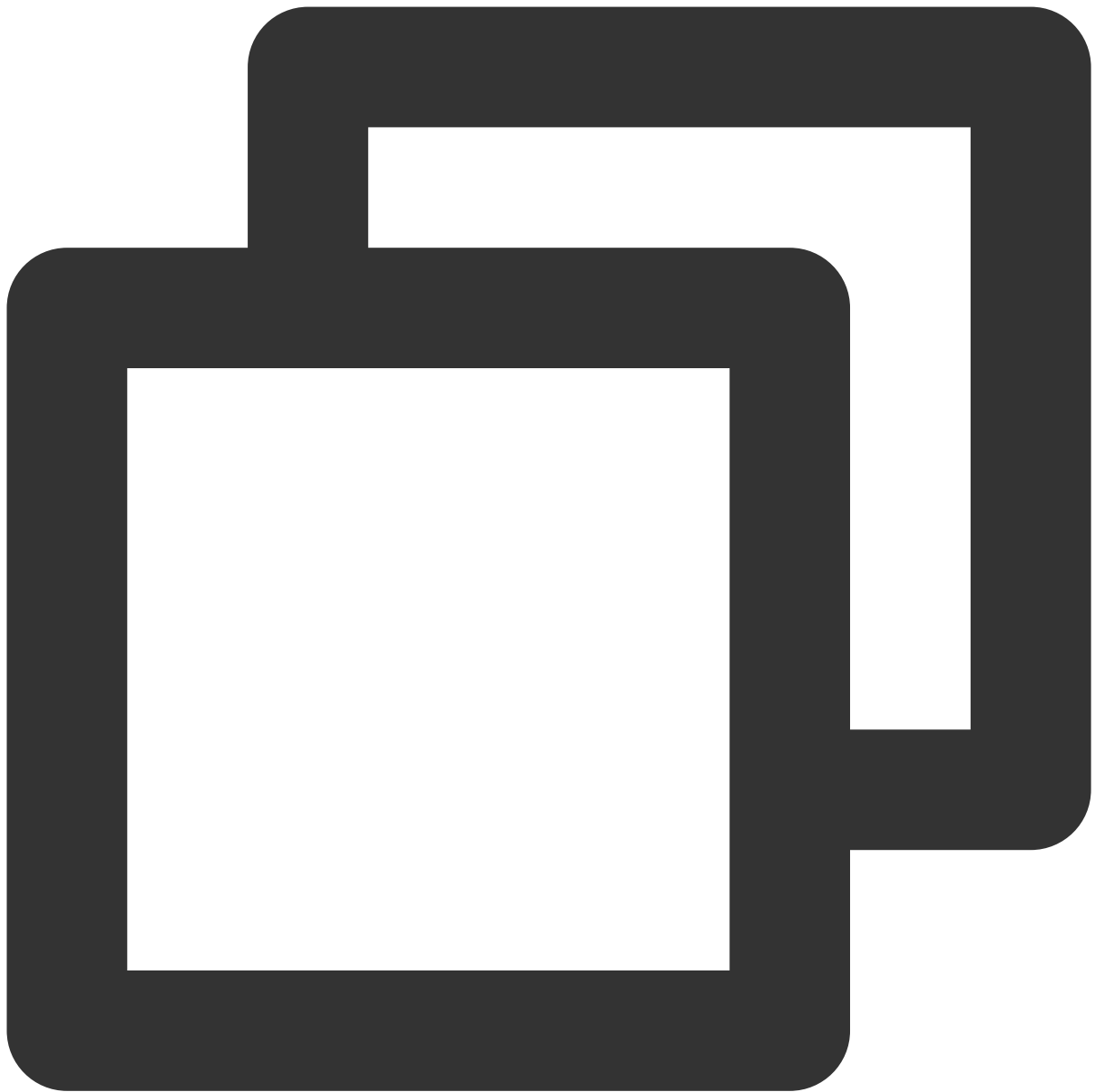
updateLoadingView方法的返回值是**IMiniLoading**类型的实例。

IMiniLoading的方法说明：

create 创建一个 Loading 视图；

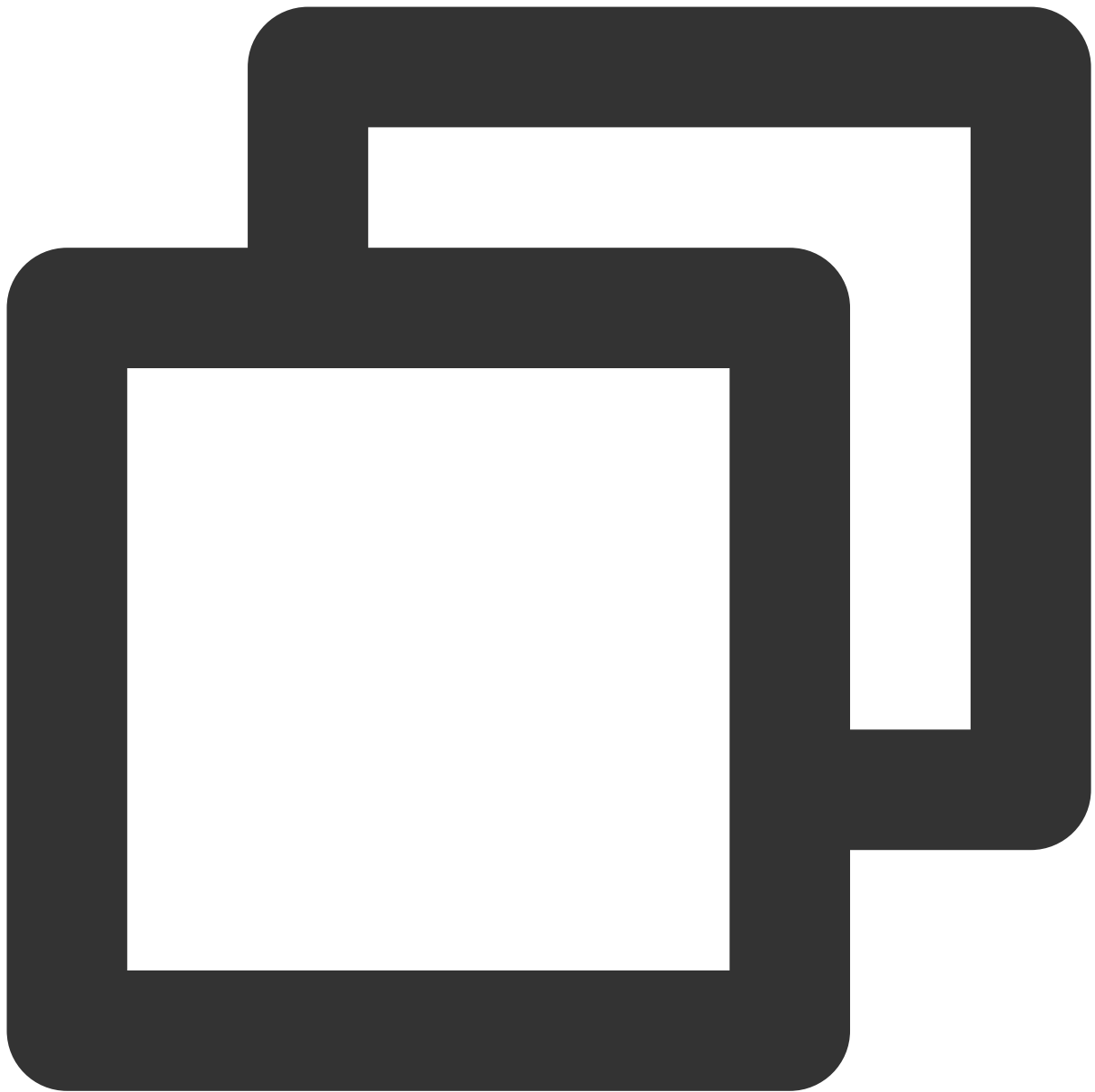
show 展示 Loading 效果时回调；

stop 停止 Loading 效果时回调。



```
/**
 * 自定义小程序加载loading页面
 * 调用环境：子进程
 *
 * @param activityWeakRef Activity引用
 * @param app 小程序信息
 * @return 返回小程序loading UI
 */
public abstract IMiniLoading startLoadingView(WeakReference<Activity> activityWeakR
```

示例代码：



```
@Override
public IMiniLoading startLoadingView(Context context) {
    return new IMiniLoading() {
        @Override
        public View create() {
            return LayoutInflater.from(context).inflate(R.layout.applet_activity_cu
        }

        @Override
        public void show(View v) {
```

```
    }

    @Override
    public void stop(View v) {

    }

};
}
```

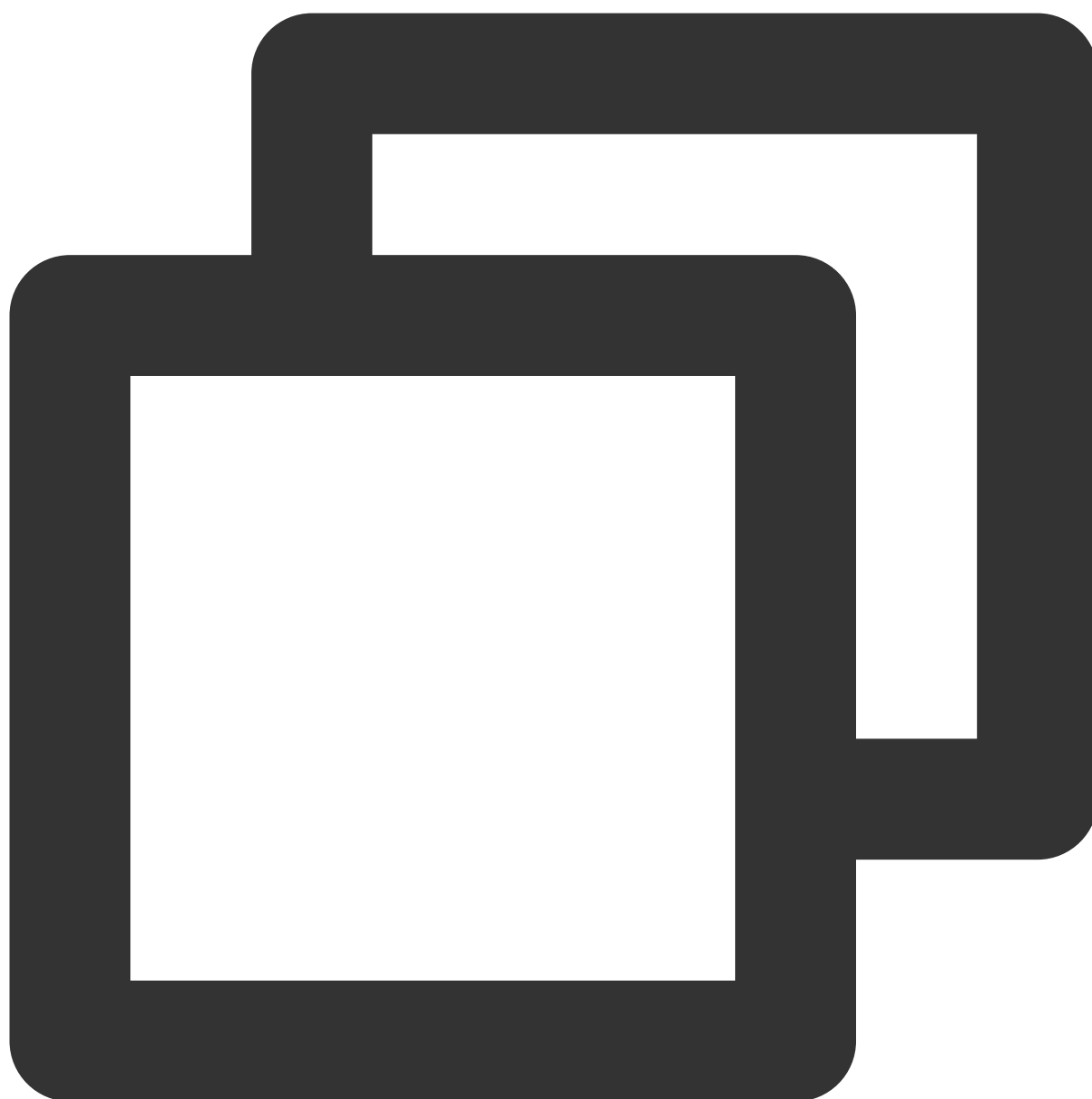
隐藏 Loading 页面的胶囊按钮

默认小程序加载页面，右上角会有胶囊按钮展示，可以通过重写 **AbsMiniUiProxy** 的 **hideLoadingCapsule** 方法控制胶囊按钮的隐藏和显示。

胶囊按钮示意图：



API说明：返回值为 **true** 表示隐藏 Loading 页面的胶囊按钮，**false** 表示不隐藏（默认值）。



```
/**
 * 是否隐藏小程序加载loading页面右上角胶囊
 *
 * @return true:隐藏;false:不隐藏（默认值）
 */
boolean hideLoadingCapsule();
```

定制小程序 SDK 接口

最近更新时间：2024-06-27 10:54:21

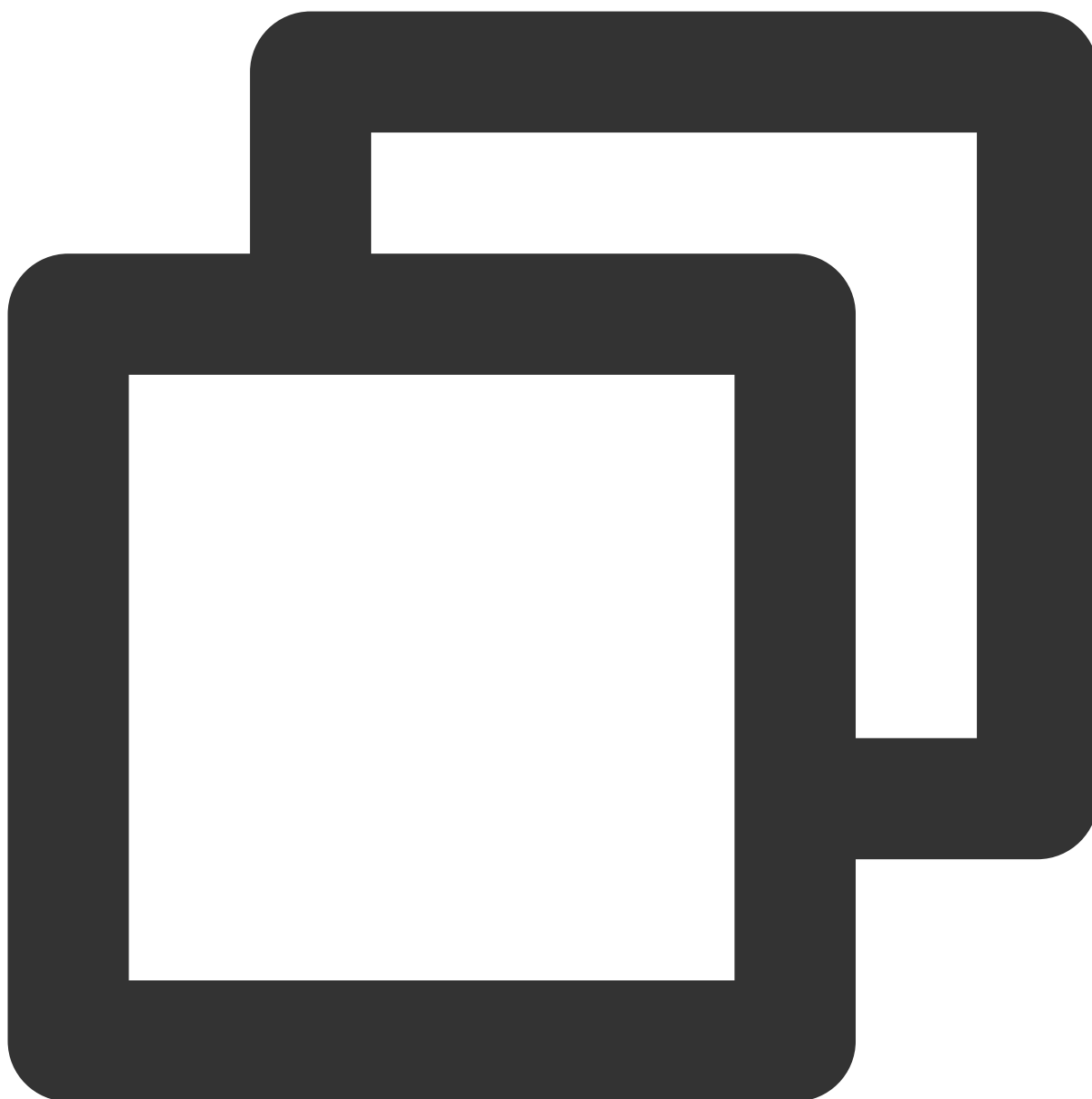
为了丰富使用场景，小程序 SDK 对外开放了一部分接口，这部分接口可以由宿主应用定制实现，以提供宿主应用特色的能力给小程序使用。

前提条件

实现小程序 SDK 的接口定制，需要按照如下的方式自定义 `MiniAppProxy`：

说明：

定义实现类并继承 `BaseMiniAppProxy`，并使用 `@ProxyService(proxy = MiniAppProxy.class)` 注解进行修饰。



```
@ProxyService(proxy = MiniAppProxy.class)
public class MiniAppProxyImpl extends BaseMiniAppProxy{}
```

前提条件的步骤完成后，可以按照下面的方法进行小程序 SDK 接口进行自定义，小程序SDK主要支持了以下几种类型的接口。

一、定制胶囊按钮功能

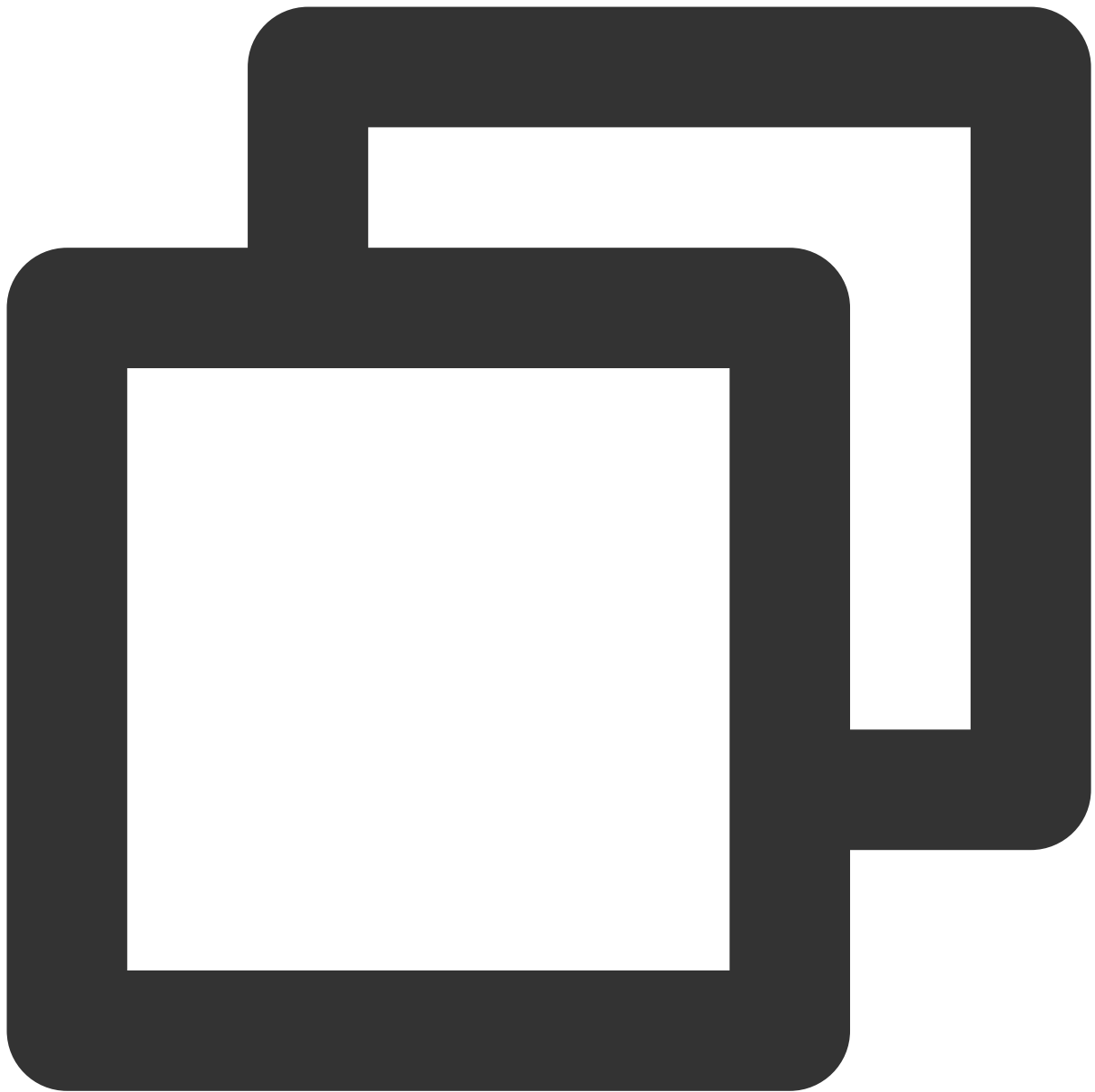
1.1 定制关闭按钮事件监听

定制胶囊按钮的关闭事件监听，能够让宿主应用在关闭按钮点击时获取到对应的回调事件。

关闭按钮示意图：

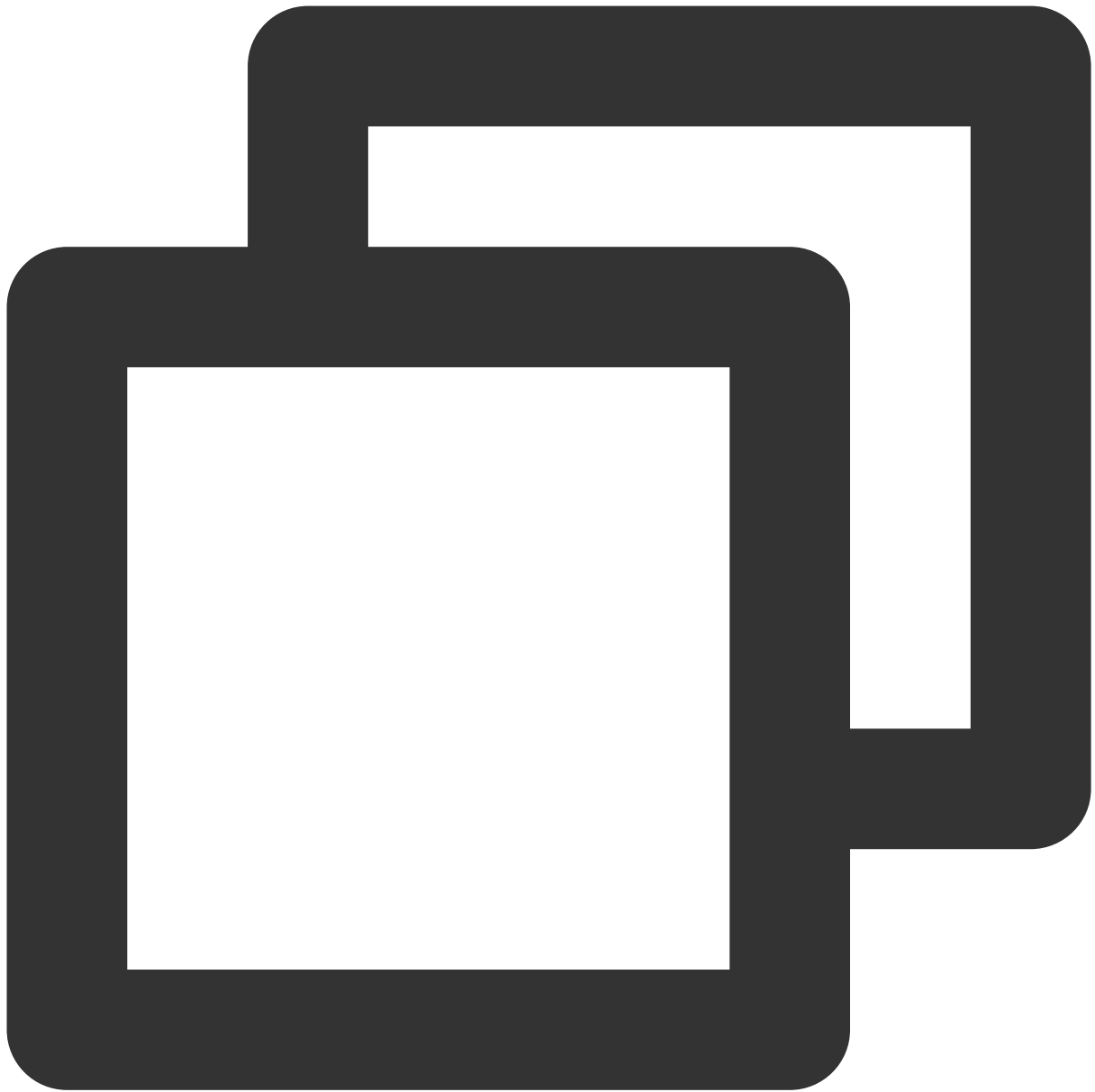


API 说明： `onCapsuleButtonCloseClick` 方法返回值表示是否自定义关闭按钮监听，返回值为 `true` 表示自定义，`false`（默认值）表示使用默认监听。



```
/**
 * 点击胶囊按钮的关闭选项
 * 调用环境：子进程
 *
 * @param miniAppContext 小程序运行环境(小程序进程，非主进程)
 * @param onCloseClickedListener 点击小程序关闭时回调
 * @return 不支持该接口，请返回false
 */
public abstract boolean onCapsuleButtonCloseClick(IMiniAppContext miniAppContext,
    DialogInterface.OnClickListener onCloseClickedListener);
```


示例代码：



```
@Override
public boolean onCapsuleButtonCloseClick(IMiniAppContext miniAppContext,
                                          DialogInterface.OnClickListener onClos
    Log.e("TAG", "onCapsuleButtonCloseClick"+miniAppContext.getMiniAppInfo());
    //handle close mini app event
    return true;
}
```

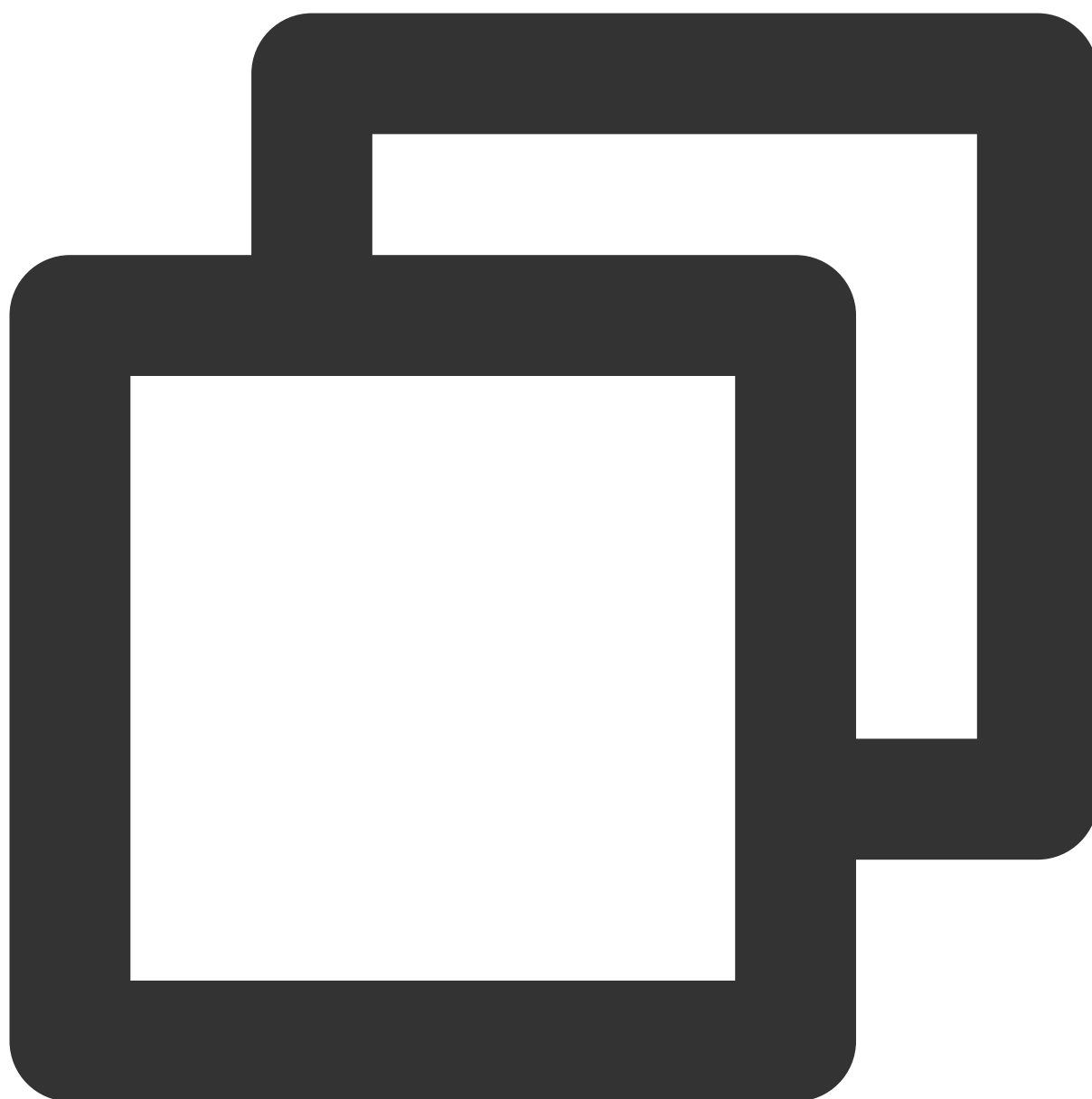
1.2 定制更多按钮事件监听

定制更多按钮的事件监听，能够让宿主应用在点击更多按钮点击时监听到到对应 item 的回调事件。

更多按钮示意图：

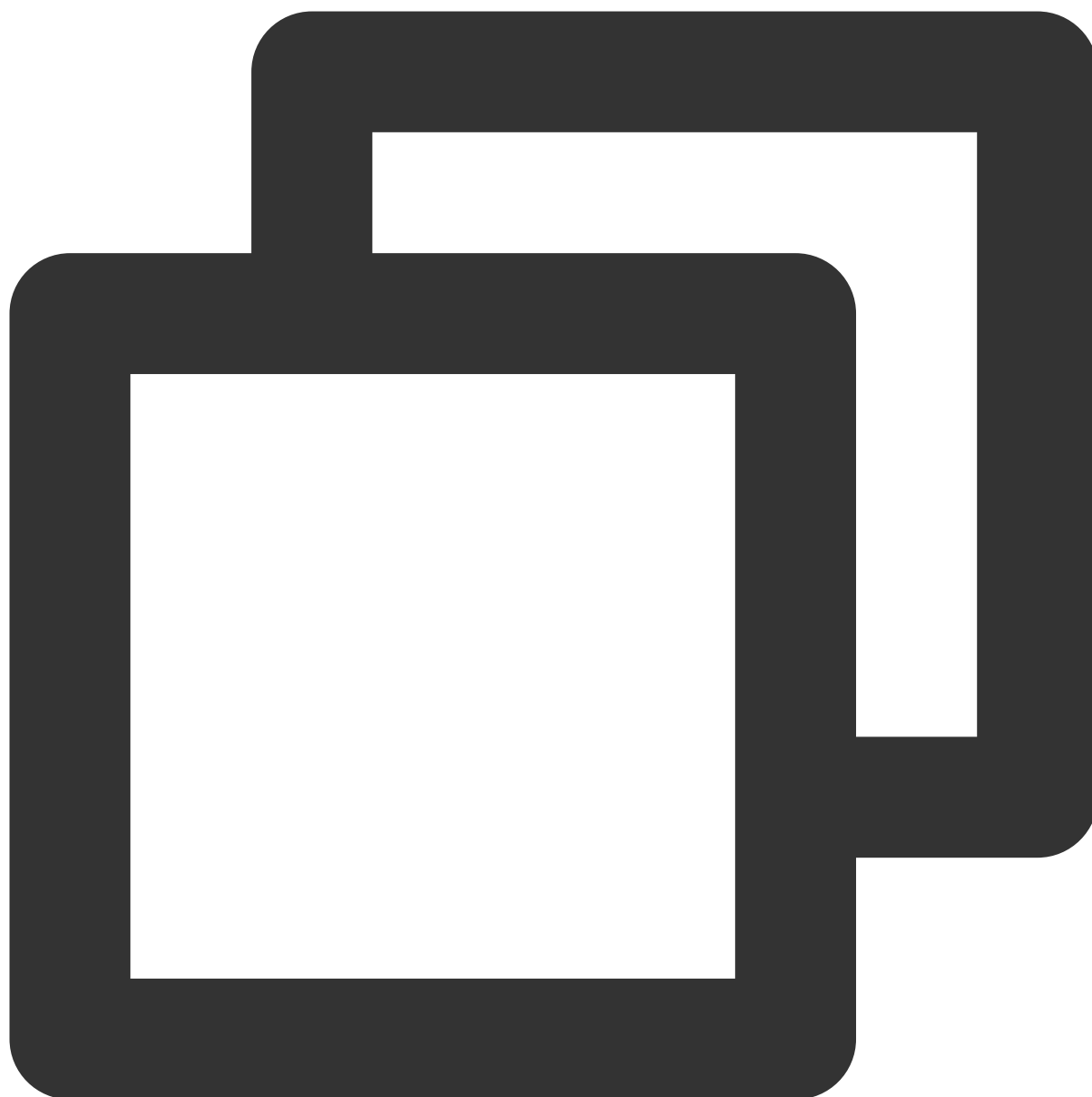


API说明：`getMoreItemSelectedListener` 方法返回值表示更多按钮监听。



```
/**
 * 返回胶囊更多面板按钮点击监听器
 * 调用环境：子进程
 *
 * @return 监听器
 */
public abstract OnMoreItemSelectedListener getMoreItemSelectedListener();
```

示例代码：



```
/**
 * 返回胶囊更多面板按钮点击监听器
 *
 * @return
 */
@Override
public OnMoreItemSelectedListener getMoreItemSelectedListener() {
    return new DemoMoreItemSelectedListener();
}

/**
```

```
define of DemoMoreItemSelectedListener
**/
public class DemoMoreItemSelectedListener extends DefaultMoreItemSelectedListener {
    public static final int CLOSE_MINI_APP = 150;

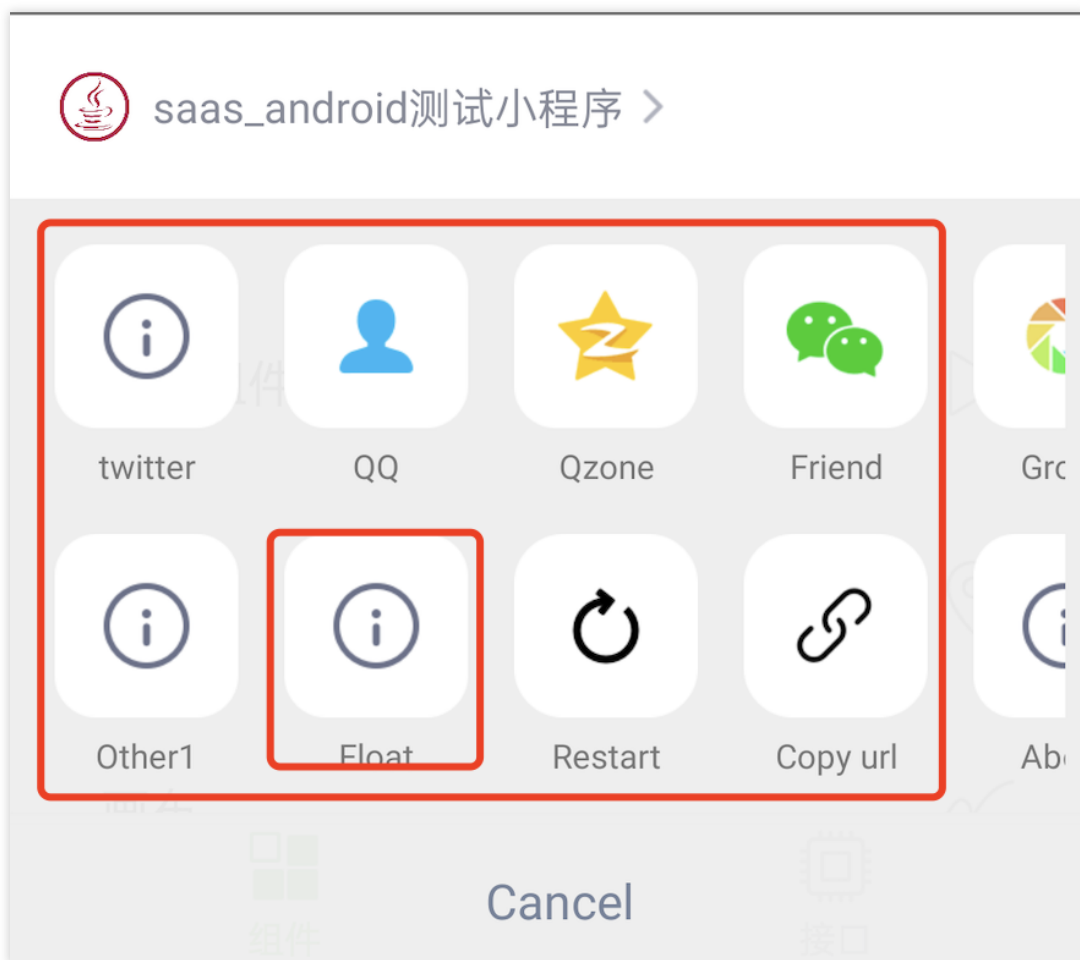
    @Override
    public void onMoreItemSelected(IMiniAppContext miniAppContext, int moreItemId)

        switch (moreItemId) {
            case CLOSE_MINI_APP:
                close(miniAppContext);
                return;
            case OTHER_MORE_ITEM_1:
                miniAppContext.getAttachedActivity().runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        Toast.makeText(miniAppContext.getAttachedActivity(), "custo
                    }
                });
                return;
        }
        super.onMoreItemSelected(miniAppContext, moreItemId);
    }

    public void close(IMiniAppContext miniAppContext) {
        Activity activity = miniAppContext.getAttachedActivity();
        if (activity != null && !activity.isFinishing()) {
            boolean moved = activity.moveTaskToBack(true);
            if (!moved) {
                QMLLog.e("Demo", "moveTaskToBack failed, finish the activity.");
                activity.finish();
            }
        }
    }
}
```

1.3 定制更多按钮展示列表

到用户触发更多按钮点击事件时，会弹出如下的可选扩展按钮列表，默认列表示意图如下：



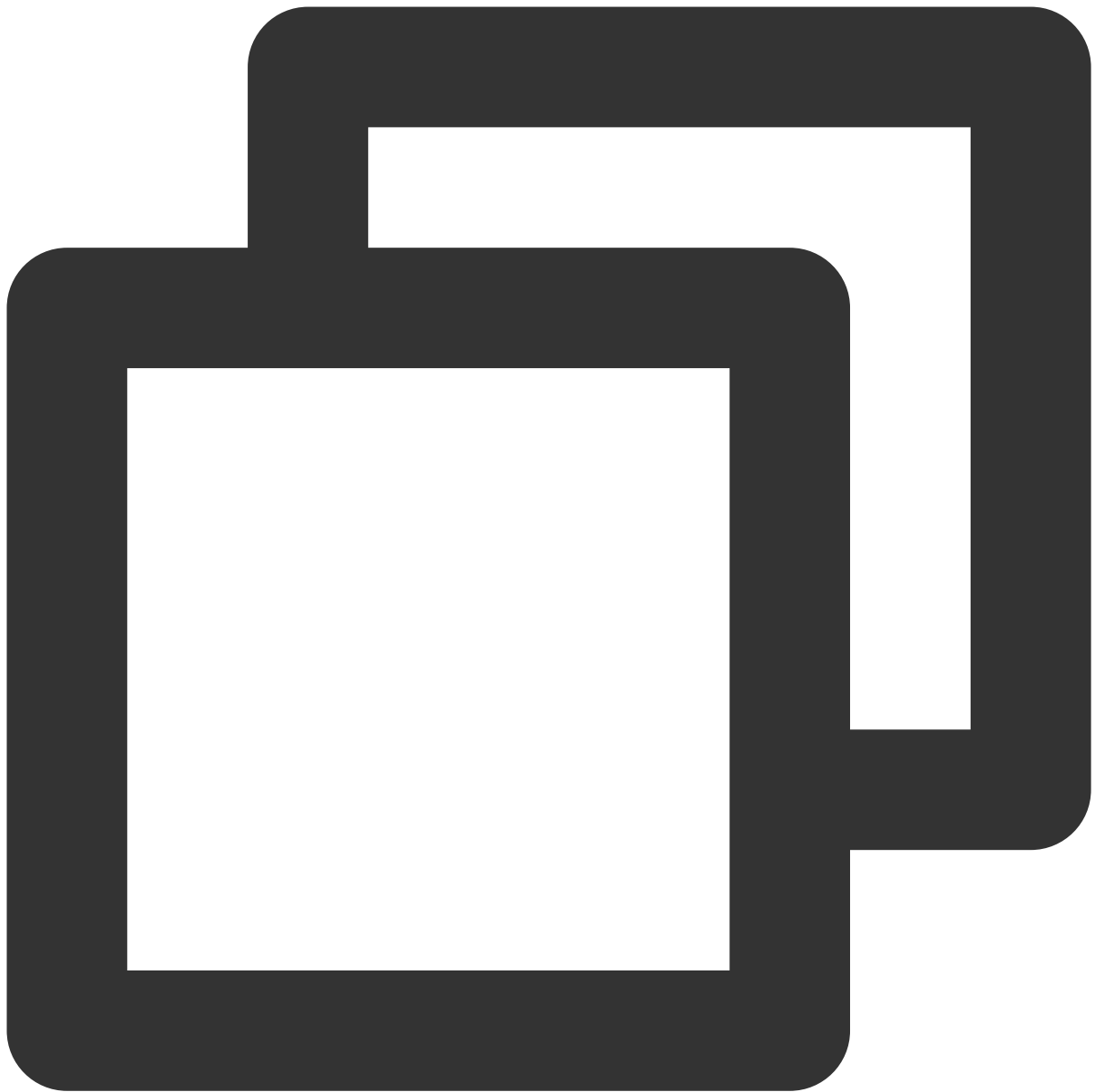
通过重写 `MiniAppProxy` 的 `getMoreItems` 方法，可以实现定制更多菜单扩展按钮列表。

API 说明：

`getMoreItems` 方法返回值表示定制的扩展按钮列表；

方法参数 `MoreItemList.Builder` 用于添加扩展按钮，扩展按钮的展示顺序和添加顺序一致；

`MoreItem` 的 `id` 属性用于区分按钮，需要具备唯一性。

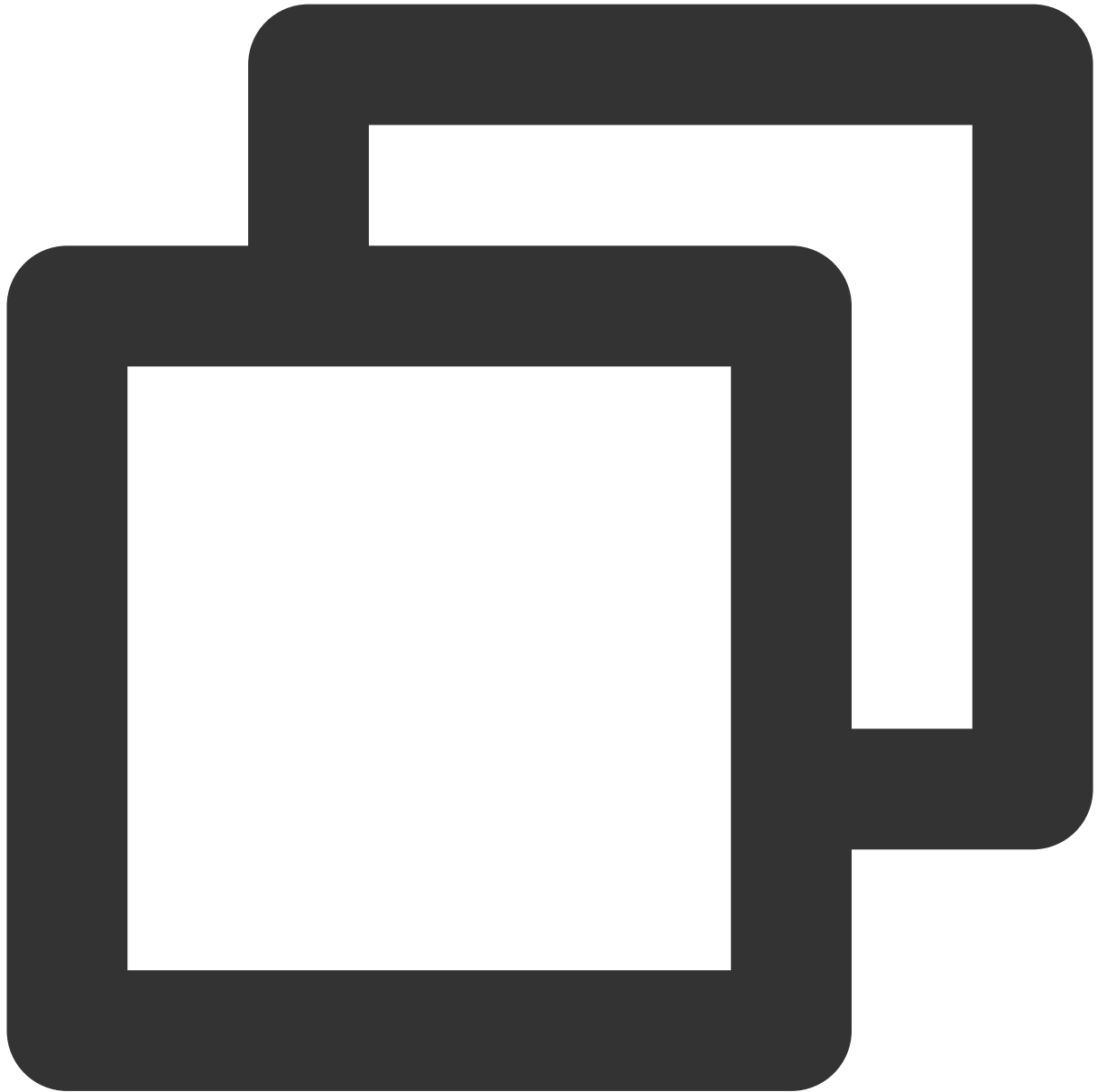


```
/**
 * 返回胶囊更多面板的按钮，扩展按钮的ID需要设置为[100, 200]这个区间中的值，否则，添加无效
 * 调用环境：子进程
 *
 * @param builder
 * @return
 */
public abstract ArrayList<MoreItem> getMoreItems(MoreItemList.Builder builder);
```

警告：

扩展按钮的ID需要设置为[100, 200]这个区间中的值，否则，添加无效。

示例代码：



```
@Override
public ArrayList<MoreItem> getMoreItems(IMiniAppContext miniAppContext, MoreItemLis
    MoreItem item1 = new MoreItem();
    item1.id = ShareProxyImpl.OTHER_MORE_ITEM_1;
    item1.text = getString(miniAppContext, R.string.applet_mini_proxy_impl_other1);
    item1.drawable = R.mipmap.mini_demo_about;

    MoreItem item2 = new MoreItem();
```



```
item2.id = ShareProxyImpl.OTHER_MORE_ITEM_2;
item2.text = getString(miniAppContext, R.string.applet_mini_proxy_impl_other2);
item2.shareKey = SHARE_TWITTER; //自定义分享的key,必须设置且唯一,与小程序端调用控制设置
item2.drawable = R.mipmap.mini_demo_about;

MoreItem item3 = new MoreItem();
item3.id = DemoMoreItemSelectedListener.CLOSE_MINI_APP;
item3.text = getString(miniAppContext, R.string.applet_mini_proxy_impl_float_ap
item3.drawable = R.mipmap.mini_demo_about;

MoreItem item4 = new MoreItem();
item4.id = ShareProxyImpl.OTHER_MORE_ITEM_INVALID;
item4.text = getString(miniAppContext, R.string.applet_mini_proxy_impl_out_of_e
item4.drawable = R.mipmap.mini_demo_about;

// 自行调整顺序。
builder.addMoreItem(item1)
    .addMoreItem(item2)
    .addShareQQ("QQ", R.mipmap.mini_demo_channel_qq)
    .addMoreItem(item3)
    .addShareQzone(getString(miniAppContext, R.string.applet_mini_proxy_imp
        R.mipmap.mini_demo_channel_qzone)
    .addShareWxFriends(getString(miniAppContext, R.string.applet_mini_proxy
        R.mipmap.mini_demo_channel_wx_friend)
    .addShareWxMoments(getString(miniAppContext, R.string.applet_mini_proxy
        R.mipmap.mini_demo_channel_wx_moment)
    .addRestart(getString(miniAppContext, R.string.applet_mini_proxy_impl_r
        R.mipmap.mini_demo_restart_miniapp)
    .addAbout(getString(miniAppContext, R.string.applet_mini_proxy_impl_abo
        R.mipmap.mini_demo_about)
    .addDebug(getString(miniAppContext, R.string.mini_sdk_more_item_debug),
        R.mipmap.mini_demo_about)
    .addMonitor(getString(miniAppContext, R.string.applet_mini_proxy_impl_p
        R.mipmap.mini_demo_about)
    .addComplaint(getString(miniAppContext, R.string.applet_mini_proxy_impl
        R.mipmap.mini_demo_browser_report)
    .addSetting(getString(miniAppContext, R.string.mini_sdk_more_item_setti
        R.mipmap.mini_demo_setting);

return builder.build();
}

private String getString(IMiniAppContext miniAppContext, int id) {
    return miniAppContext.getContext().getString(id);
}
```

二、定制图片选择

小程序 SDK 提供默认的图片选择，默认使用 Android 系统自带的相册选择器进行选择，在小程序调用多媒体选择 `wx.chooseImage` 触发。

定制图片选择可以通过重写 `BaseMiniAppProxy` 的 `openChoosePhotoActivity` 方法实现。

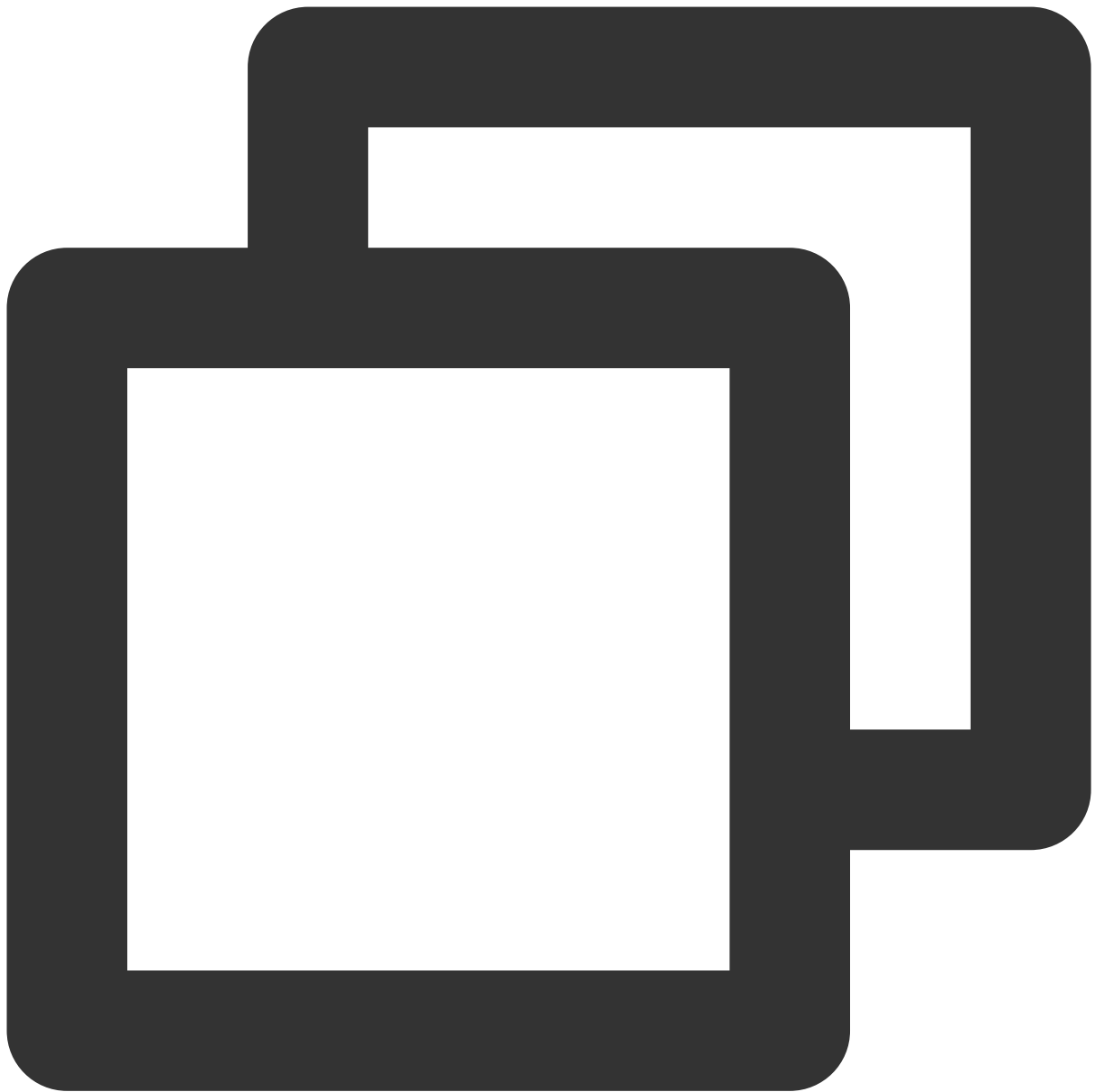
API 说明：

参数 `Context`：当前小程序页面的 `activity` 实例；

参数 `maxSelectedNum`：最大可选择图片数量；

参数 `IChoosePhotoListner`：图片选择回调接口；

返回值 `boolean`：返回值为 `false` 表示使用默认图片选择实现，返回值为 `true` 表示使用定制图片选择实现。

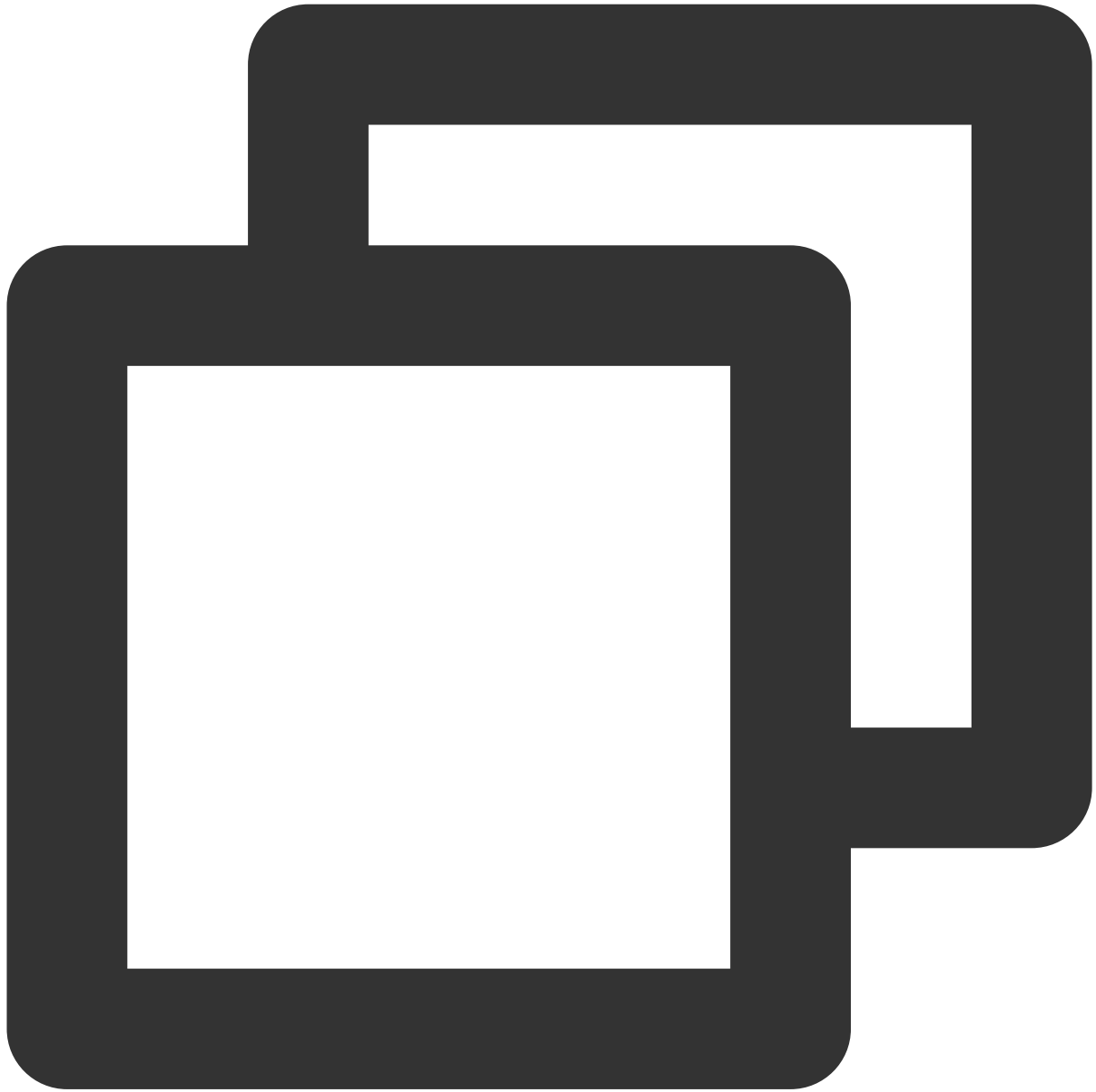


```
/**
 * 打开选图界面
 *
 * @param context      当前Activity
 * @param maxSelectedNum 允许选择的最大数量
 * @param listner      回调接口
 * @return 不支持该接口，请返回false
 */
@Override
public boolean openChoosePhotoActivity(Context context, int maxSelectedNum, IChoose
```

注意：

通过 IChoosePhotoListner 的 onResult 方法返回图片路径时，图片路径应当为绝对路径。

示例代码：



```
@Override
public boolean openChoosePhotoActivity(Context context, int maxSelectedNum, IChoose
    Log.d("TAG", "open choose photo activity ");
    Intent intent = new Intent(context, ChoosePhotosActivity.class);
    intent.putExtra("maxCount", maxSelectedNum);
    //not recommend to use static refs, just for example
    ChoosePhotosActivity.setChooseCallBack(listner);
```

```
context.startActivity(intent);

return true;
}

public class ChoosePhotosActivity extends Activity {
    static WeakReference<MiniAppProxy.IChoosePhotoListner>iChoosePhotoListnerWeakRef
    public static void setChooseCallBack(MiniAppProxy.IChoosePhotoListner choosePho
        iChoosePhotoListnerWeakReference = new WeakReference<>(choosePhotoListner);
    }

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //todo show your choose view
    }

    @Override
    protected void onResume() {
        super.onResume();
        startChooseImage();
    }

    private void startChooseImage(){
        //todo finish choose logic

        //response choose callback
        ArrayList<String>path = new ArrayList<>();
        //todo replace with real path of image
        path.add("picture local absolute path");
        path.add("picture local absolute path2");
        path.add("/data/user/0/com.tencent.tmf.miniapp.demo/files/userfiles/Screens
        iChoosePhotoListnerWeakReference.get().onResult(path);
    }
}
```

三、定制图片预览

小程序 SDK 提供默认的图片预览实现，在小程序调用多媒体选择 `wx.previewImage` 时触发。

定制图片预览可以通过重写 **BaseMiniAppProxy** 的 **openImagePreview** 方法实现。

默认图片预览页面如下图：



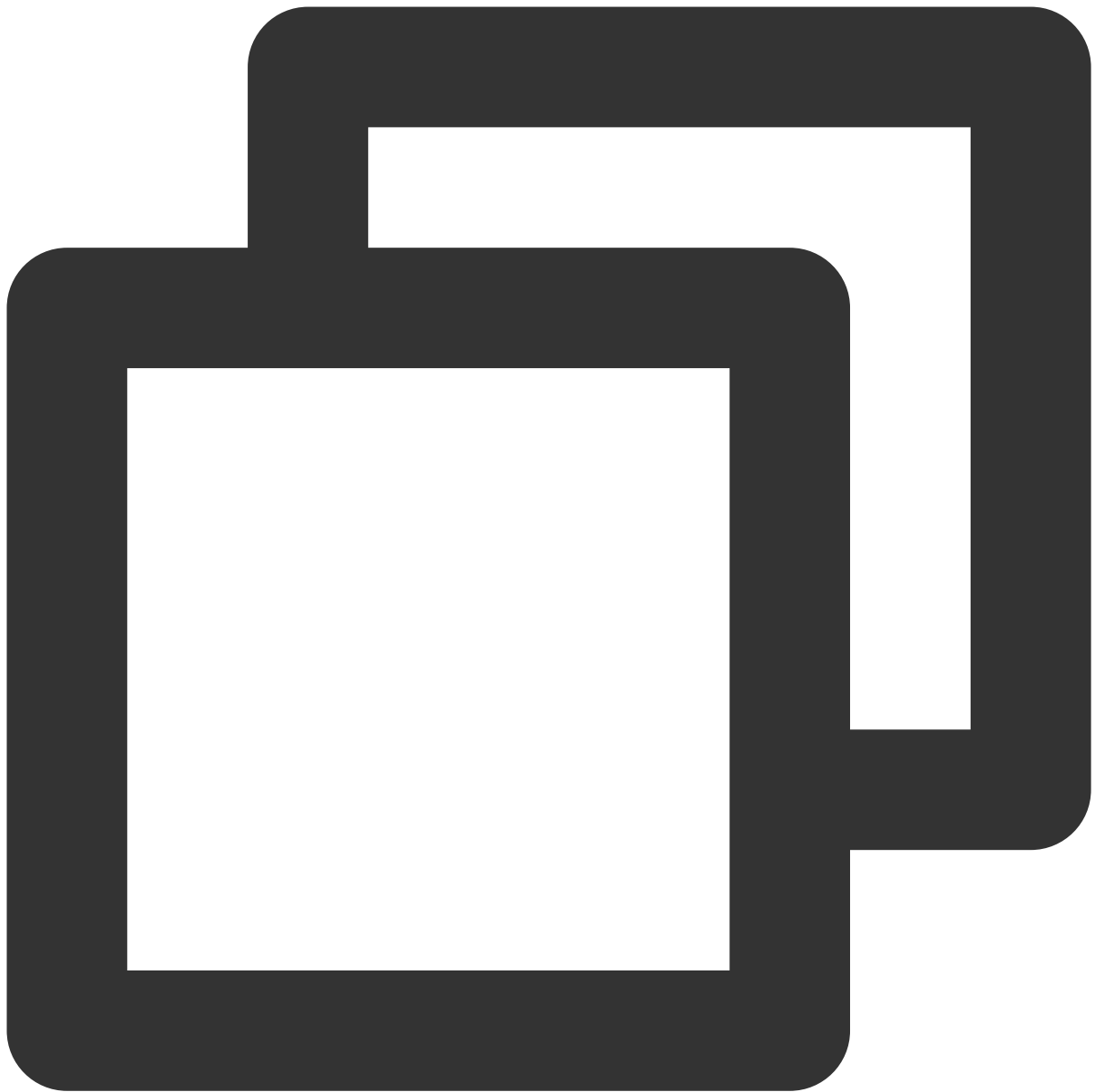
API说明：

参数 Context：当前小程序页面的 activity 实例；

参数 selectedIndex：当前选中图片在列表中位置；

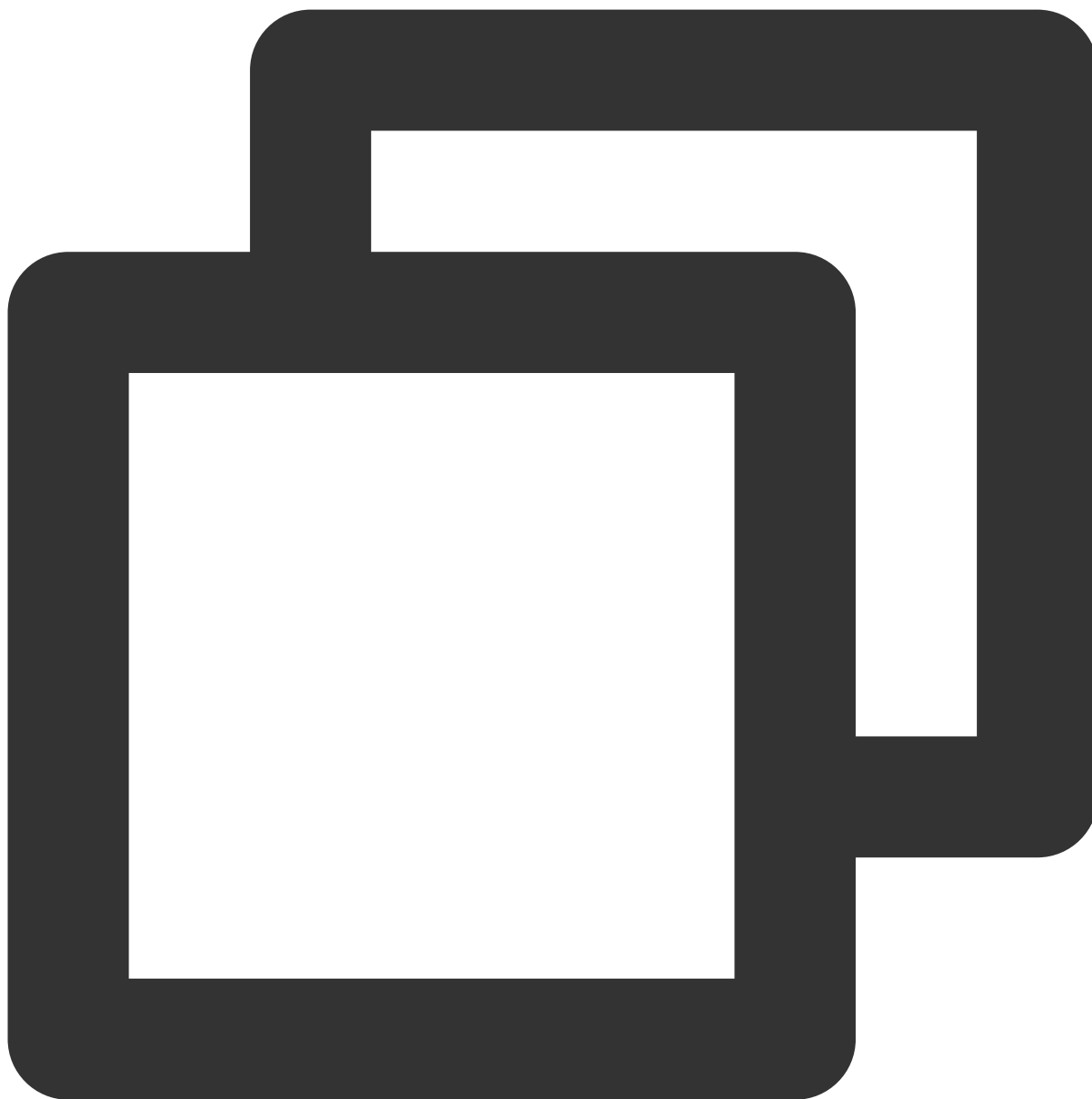
参数 pathList：待展示图片路径列表；

返回值 boolean：返回值为 false 表示使用默认图片展示，返回值为 true 表示使用定制图片展示。



```
/**
 * 打开图片预览界面
 * 调用环境：子进程
 *
 * @param context 当前Activity
 * @param selectedIndex 当前选择的图片索引
 * @param pathList 图片路径列表
 * @return 不支持该接口，请返回false
 */
public abstract boolean openImagePreview(Context context, int selectedIndex, List<S
```

示例代码：



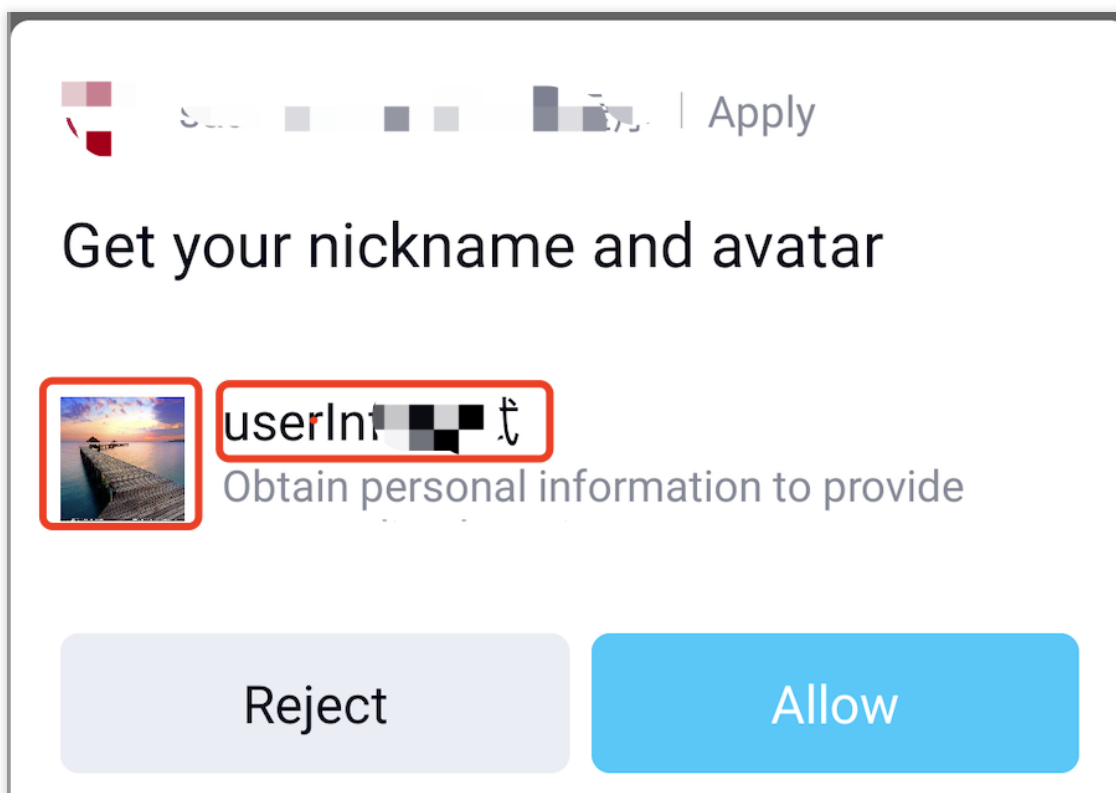
```
/**
 * 打开图片预览界面
 *
 * @param context      当前Activity
 * @param selectedIndex 当前选择的图片索引
 * @param pathList     图片路径列表
 * @return 不支持该接口，请返回false
 */
@Override
```



```
public boolean openImagePreview(Context context, int selectedIndex, List<String> pa
//todo start your image preview
Intent intent = new Intent(context, CustomPreviewActivity.class);
intent.putExtra("curIndex", selectedIndex);
intent.putStringArrayListExtra("pathList", pathList);
context.startActivity(intent);
return true;
}
```

四、定制授权弹框中的用户信息

小程序申请获取用户信息的时候，会弹出用户授权弹窗；宿主应用可以定制用户授权弹窗部分信息，包括有：头像图片和用户昵称，如下图：

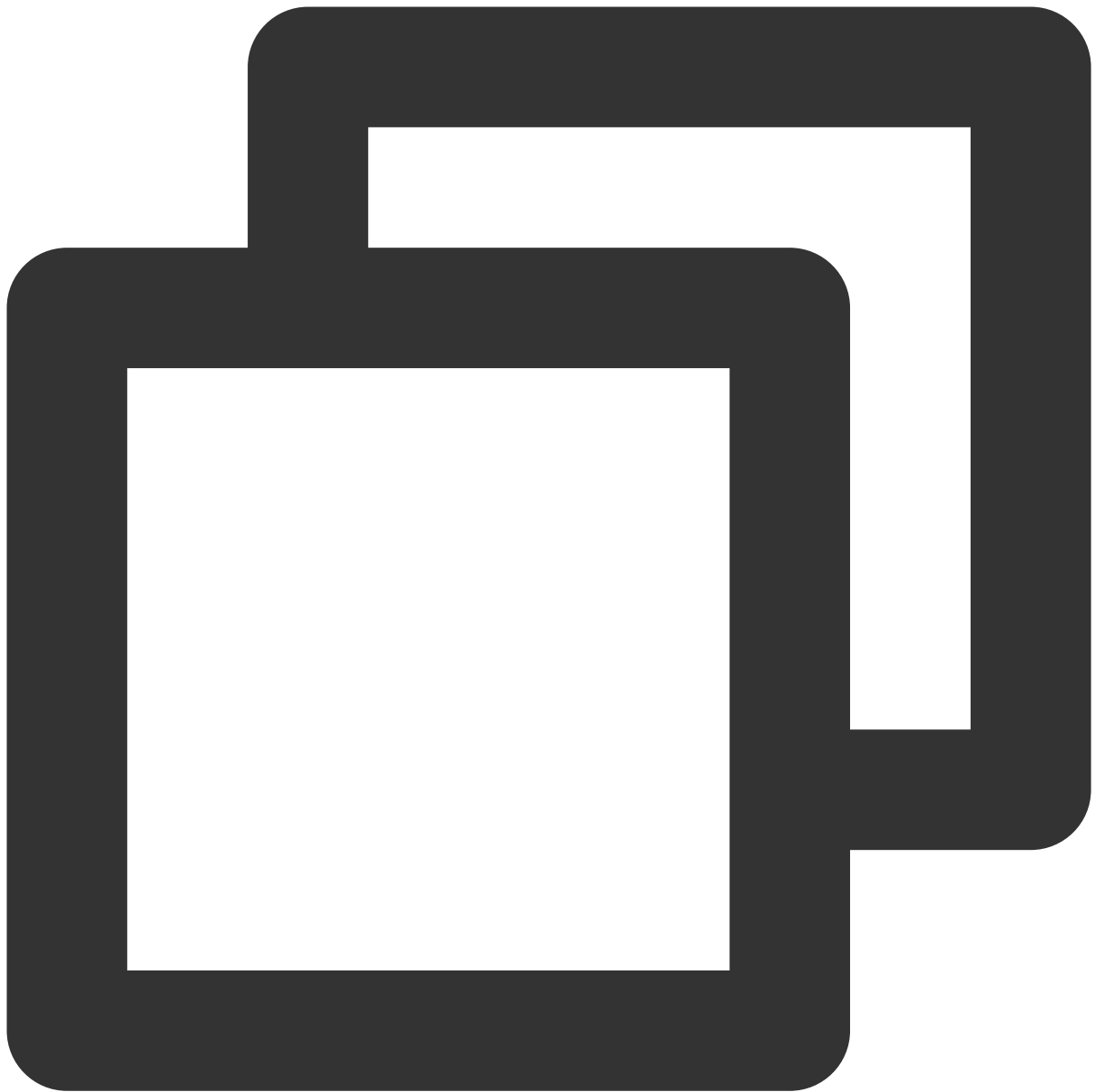


可以通过重写 **BaseMiniAppProxy** 的 **getUserInfo** 方法实现定制。

API说明：

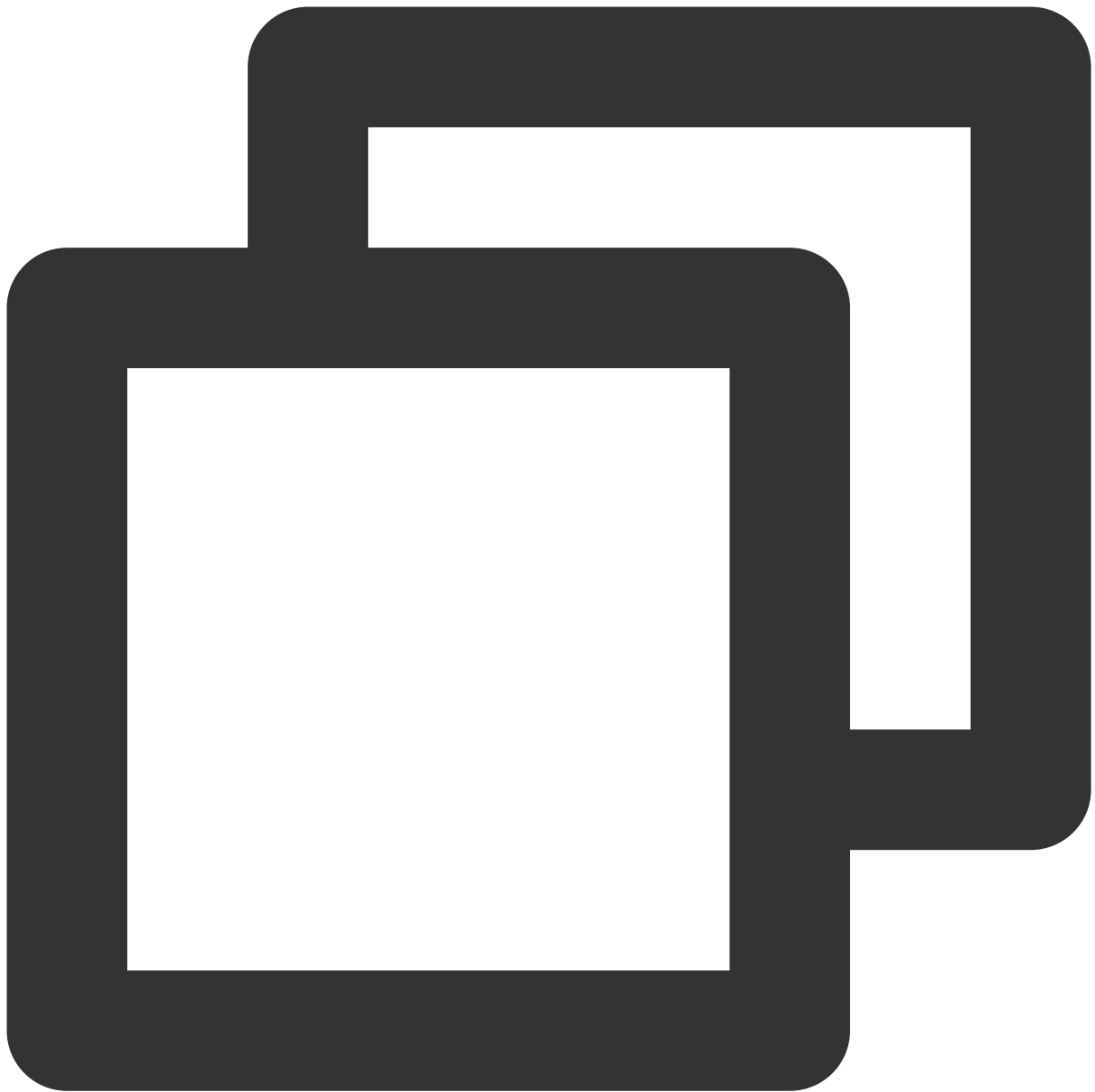
参数 **appId**：当前请求授权的小程序 **appId**；

参数 **AsyncResult**：用于返回用户信息给小程序 SDK。



```
/**
 * 获取scope.userInfo授权用户信息
 * 调用环境：子进程
 *
 * @param appId
 * @param result
 */
@Override
public void getUserInfo(String appId, AsyncResult result)
```

示例代码：



```
/**
 * 获取scope.userInfo授权用户信息
 * 调用环境：子进程
 *
 * @param appId
 * @param result
 */
@Override
public void getUserInfo(String appId, AsyncResult result) {
    JSONObject jsonObject = new JSONObject();
    try {
```

```
//返回昵称
jsonObject.put("nickName", "userInfo测试");
//返回头像url
jsonObject.put("avatarUrl", "https://gimg2.baidu.com/image_search/src=http%
result.onReceiveResult(true, jsonObject);
} catch (JSONException e) {
    e.printStackTrace();
}
}
```

五、定制图片加载器

警告：

由于小程序 SDK 没有默认的图片加载器实现，所以图片的加载依赖于宿主应用的实现；若宿主应用未实现图标加载器，则会导致部分页面的图片加载异常。

宿主应用开发者，需要通过**重写 BaseMiniAppProxy 的 getDrawable 方法实现图片加载的定制**。

API 说明：

参数 context：当前请求授权的小程序进程上下文；

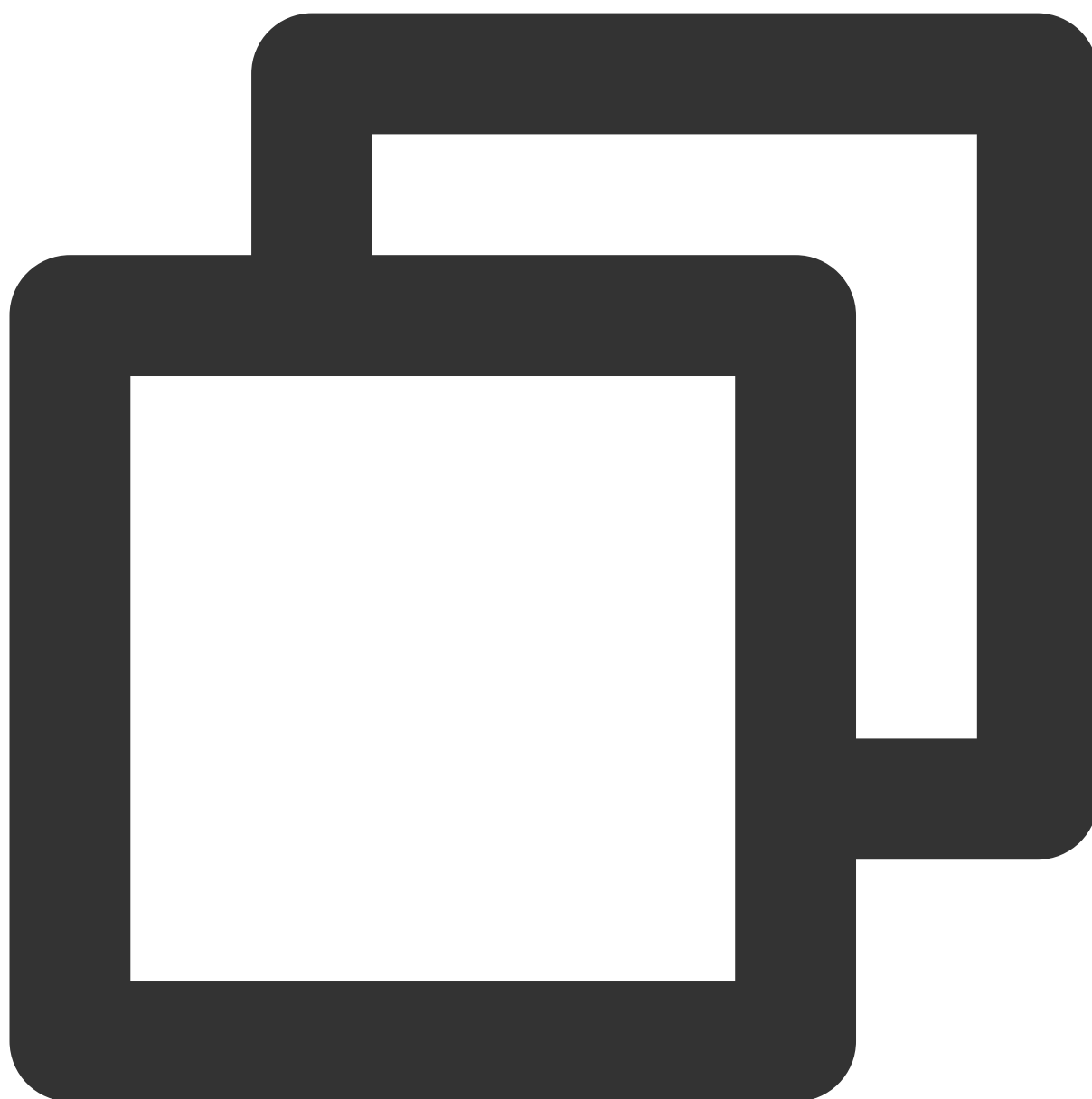
参数 source：图片来源，可以是本地或者网络图片；

参数 width：图片宽度；

参数 height：图片高度；

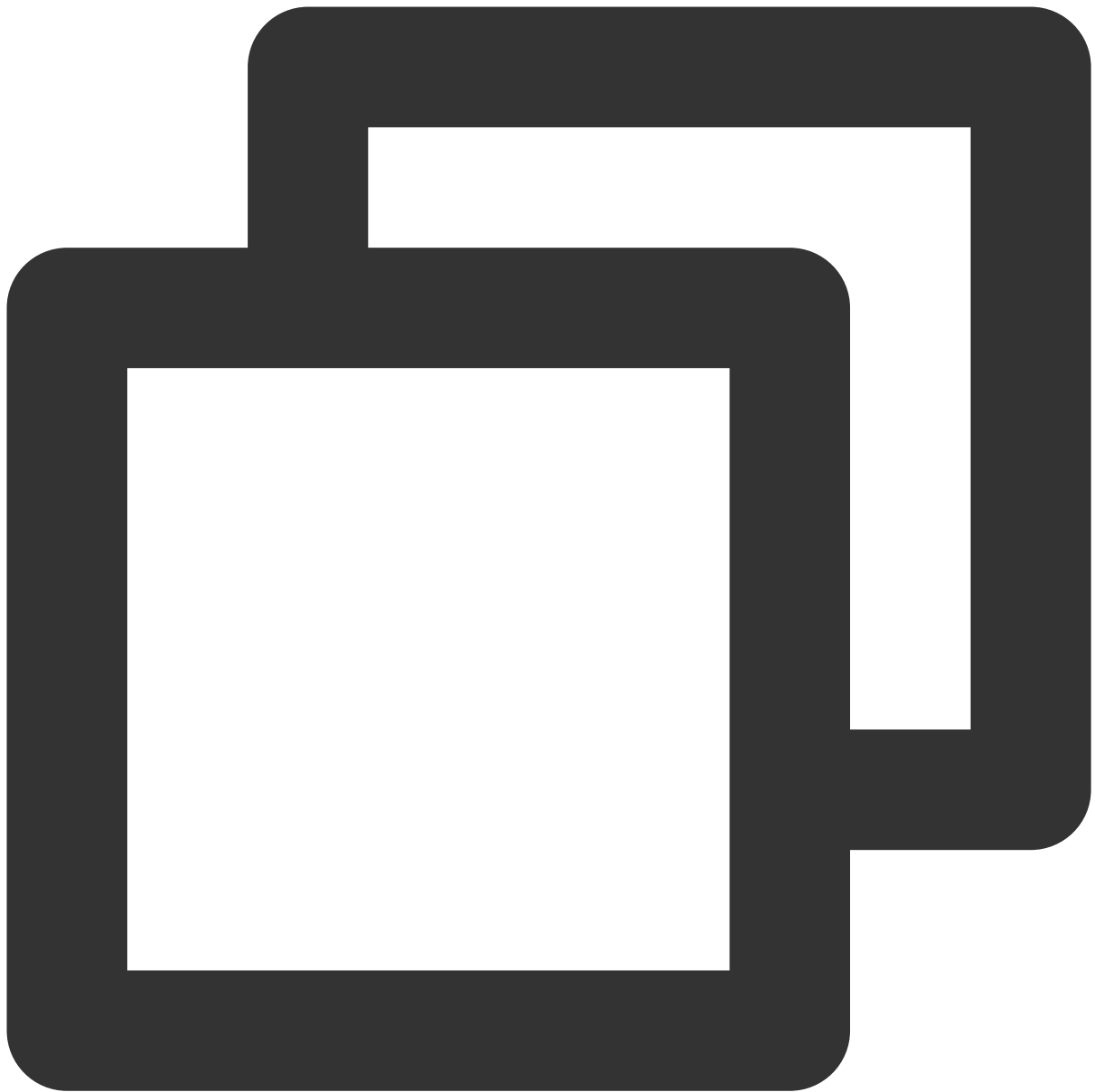
参数 defaultDrawable：默认图片，用于加载中和加载失败；

返回值 Drawable：加载的 drawable 对象。



```
@Override  
public Drawable getDrawable(Context context, String source, int width, int height, D
```

示例代码：

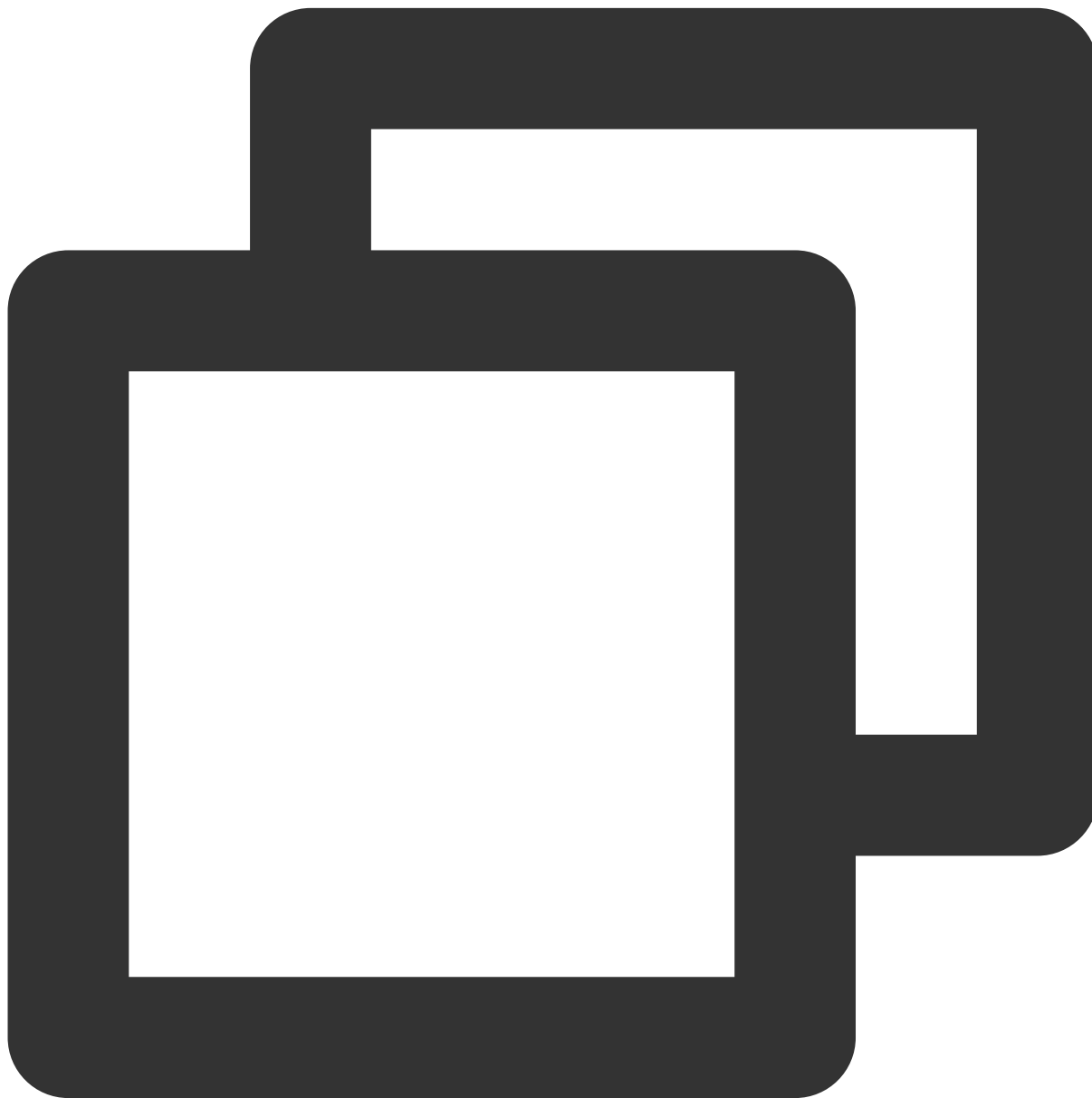


```
@Override
public Drawable getDrawable(Context context, String source, int width, int height, D
    //接入方接入自己的ImageLoader
    //示例代码里使用开源的universalimageloader
    UniversalDrawable drawable = new UniversalDrawable();
    if (TextUtils.isEmpty(source)) {
        return drawable;
    }
    drawable.loadImage(context, source);
    return drawable;
}
```

六、定制小程序数据按账户隔离存储

通过重写 **BaseMiniAppProxy** 的 **getAccount** 方法实现小程序数据的隔离存储。

API 说明：返回值是用户账号（必须唯一），设置后数据会按账号隔离存储,不建议使用用户敏感信息。



```
/**
```

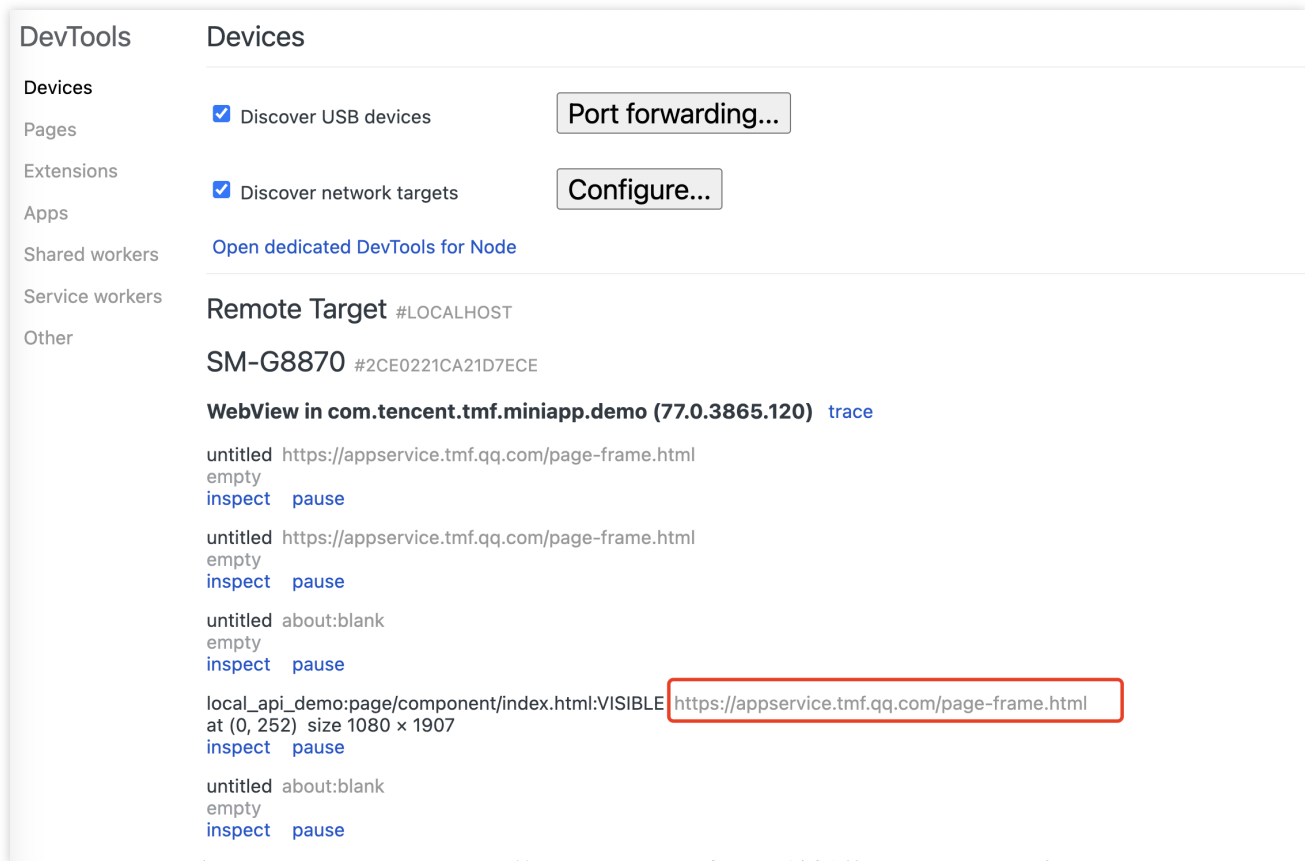
- * 用户账号, 必须唯一, 设置后数据会按账号隔离存储, 不建议使用用户敏感信息
- * 调用环境: 主进程

```
*/
@Override
public String getAccount() {
    return "tmf_test";
}
```

七、定制虚拟域名

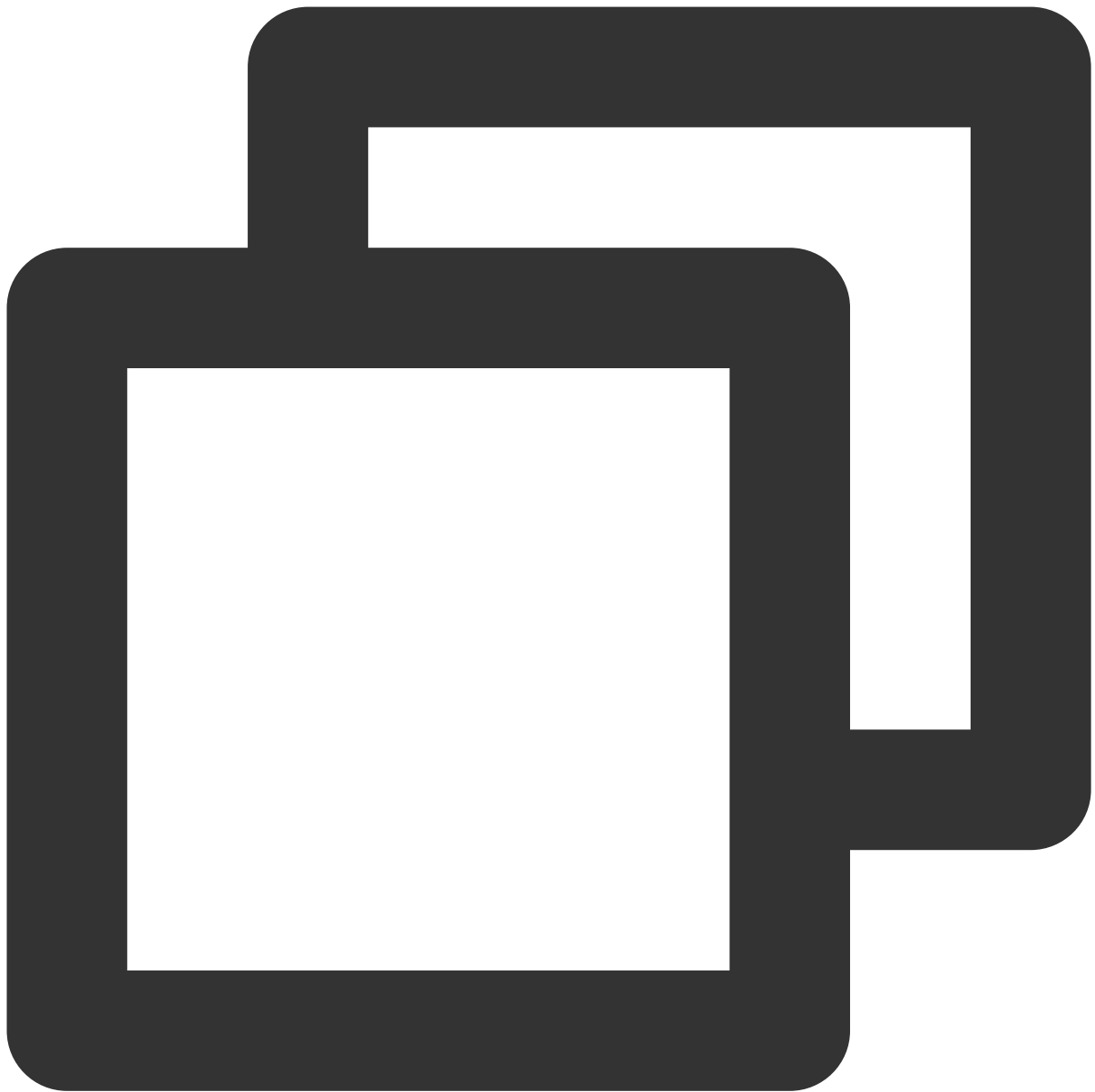
小程序打用到默认的虚拟域名，这个域名并非真实的域名，在浏览器中是无法访问的，如果需要自定义可以按照如下配置进行设置。

默认虚拟域名如下图：



定制虚拟域名，需要重写 **BaseMiniAppProxy** 的 **configData** 方法，并拦截 **configType** 为 **MiniConfigData.TYPE_DOMAIN** 的方法实现自定义。

示例代码：



```
@Override
public MiniConfigData configData(Context context, int configType, JSONObject params) {
    if(configType == MiniConfigData.TYPE_DOMAIN) {
        //虚拟域名配置
        MiniConfigData.DomainConfig domainConfig = new MiniConfigData.DomainConfig(
            domainConfig.domain = "test.com";

        return new MiniConfigData
            .Builder()
            .domainConfig(domainConfig)
            .build();
    }
}
```

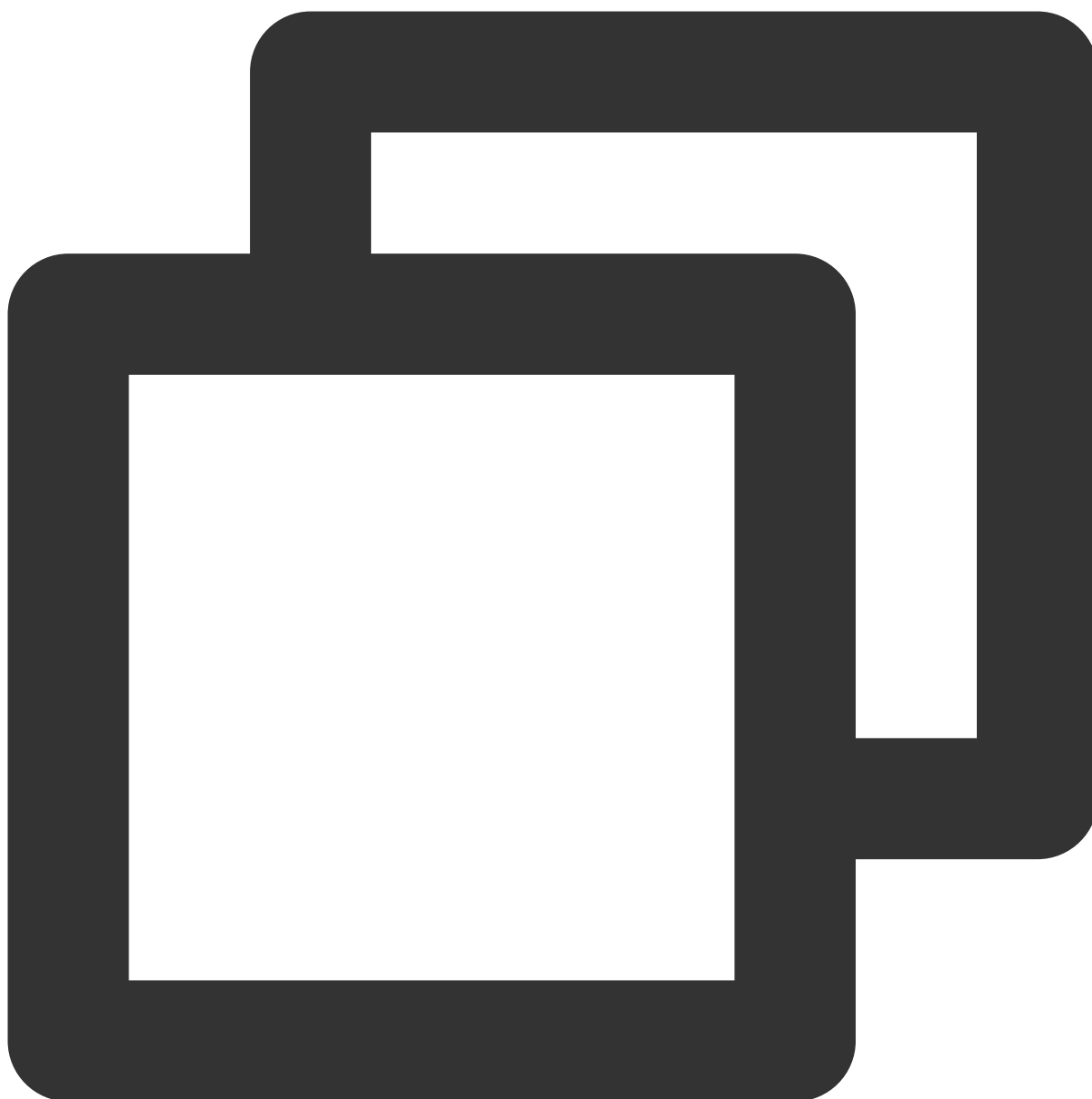
```
}

return new MiniConfigData
    .Builder()
    .build();
}
```

八、定制小程序内 WebView 的 userAgent

可以通过重写 **BaseMiniAppProxy** 的 **configData** 方法，并拦截 **configType** 为 **MiniConfigData.TYPE_WEBVIEW** 的方法实现自定义。

示例代码：



```
@Override
public MiniConfigData configData(Context context, int configType, JSONObject params) {
    if(configType == MiniConfigData.TYPE_WEBVIEW) {
        //webView userAgent
        String ua = params.optString(MiniConfigData.WebViewConfig.WEBVIEW_CONFIG_UA);
        MiniConfigData.WebViewConfig webViewConfig = new MiniConfigData.WebViewConfig();
        //设置开发者需要追加的userAgent，注意：不要把原始的ua设置进去了
        webViewConfig.userAgent = "key/value";

        return new MiniConfigData.Builder()
            .Builder()
```

```
        .webViewConfig(webViewConfig)
        .build();
    }

    return new MiniConfigData
        .Builder()
        .build();
}
```

九、定制扫码能力

小程序 SDK 提供了默认的扫描实现（参考扩展库支持-扫一扫），也对外提供接口，供用户定制扫码能力。

定制扫码能力，需要**重写 BaseMiniAppProxy 的 enterQRCode 方法实现**。

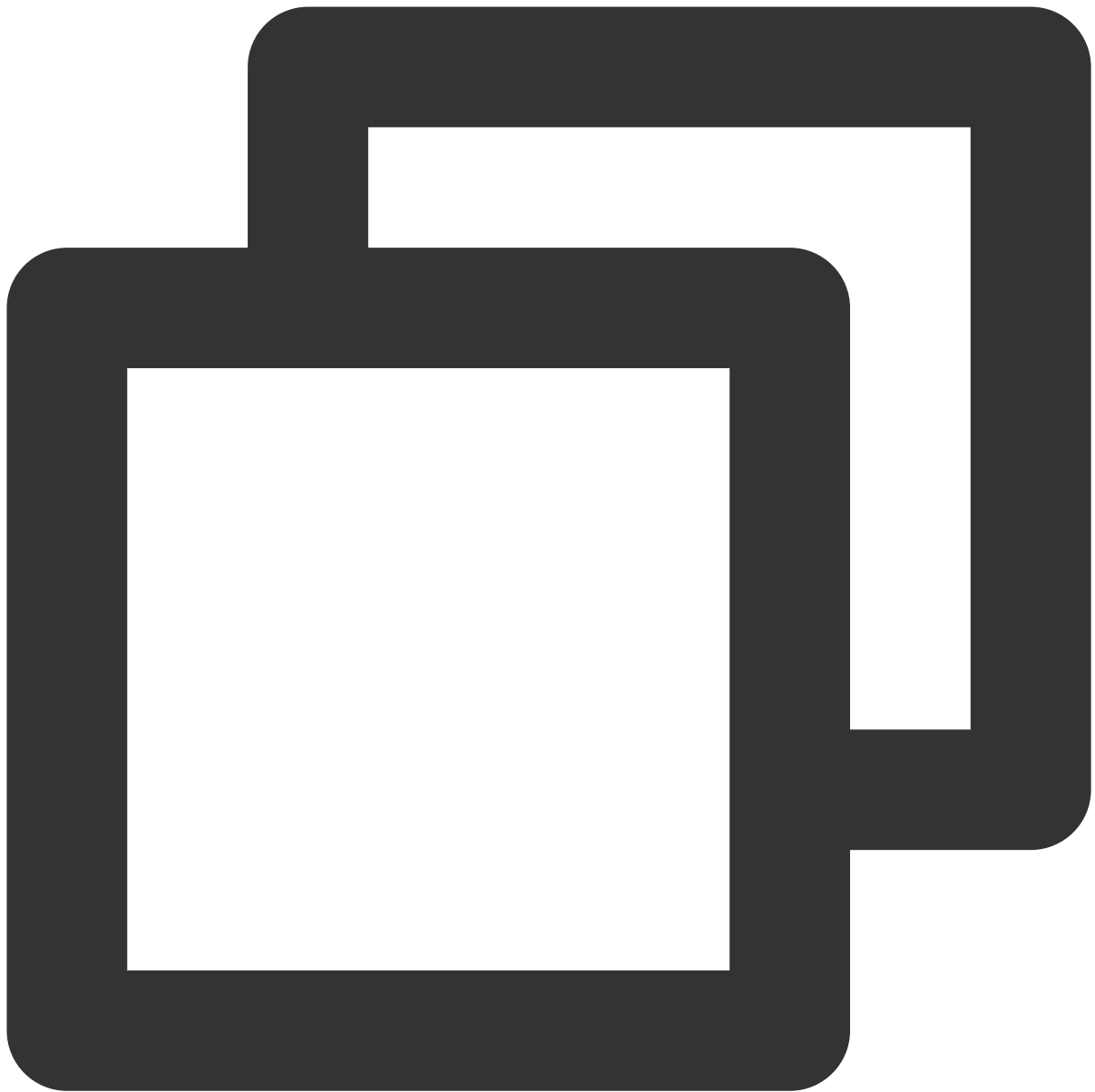
API 说明：

参数 context：当前发起扫码的小程序进程上下文；

参数 onlyFromCamera：是否只允许摄像头扫码；

参数 AsyncResult：扫码结果回调；

返回值 boolean：true 表示自定义扫码能力，false 表示使用内置扫码能力（需集成扫码扩展库）。

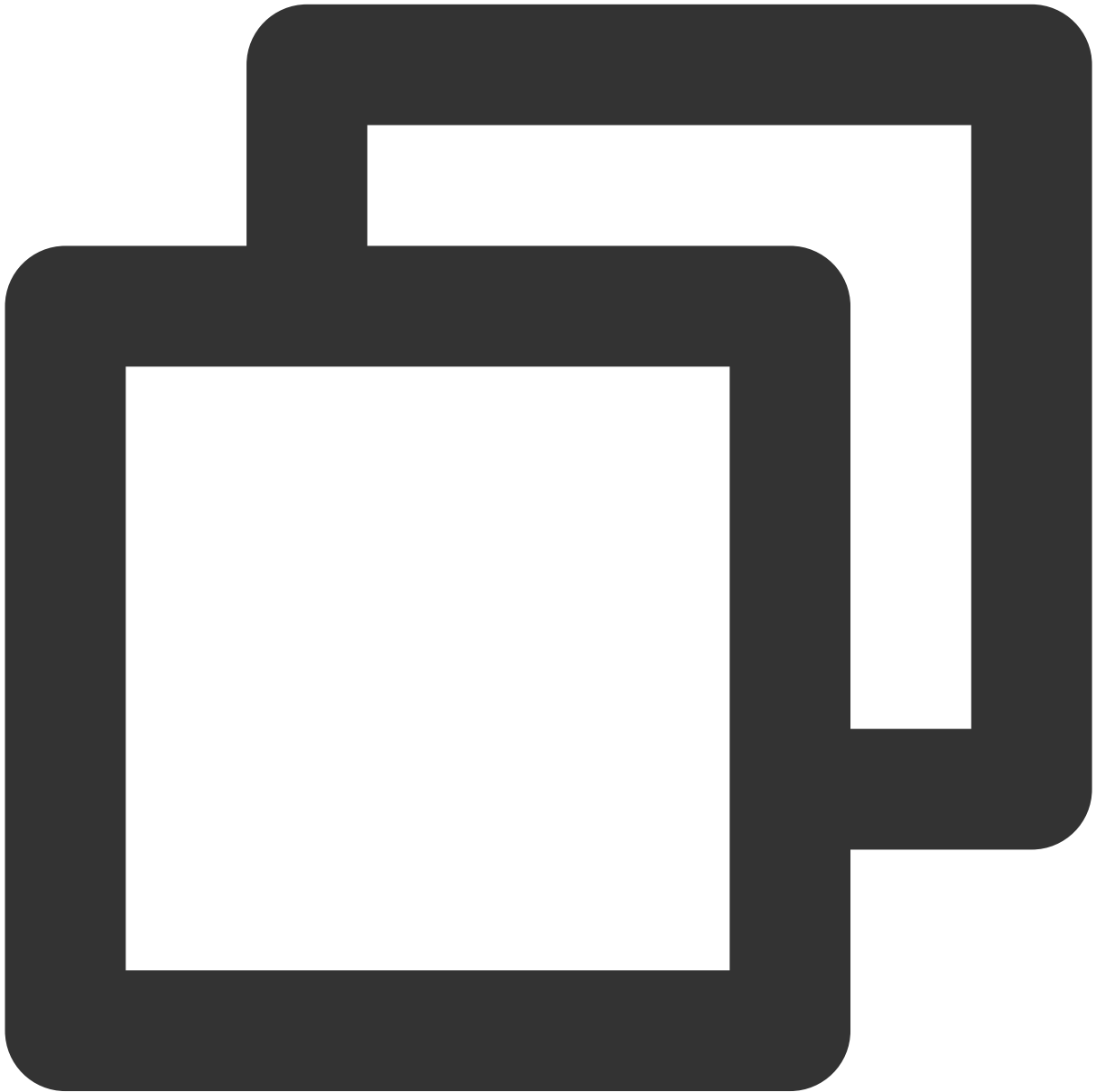


```
/**
 * 扫码二维码
 *
 * @param context      上下文
 * @param onlyFromCamera 只允许摄像头扫码
 * @param result       扫描结果
 * @return true:自定义扫码;false:使用内置扫码
 */
@Override
public boolean enterQRCode(Context context, boolean onlyFromCamera, AsyncResult res
```

十、指定小程序保存文件的存储路径

通过重写 `BaseMiniAppProxy` 的 `getSaveFileDir` 方法，实现小程序 API 保存图片、视频到系统相册中的指定目录。

示例代码：



```
/**
 * 指定小程序API保存图片、视频到系统相册中的目录
 *
 * /
```

```
public String getSaveFileDir(){  
    return "myapp";  
}
```

注意：

默认路径为 tcmpp。

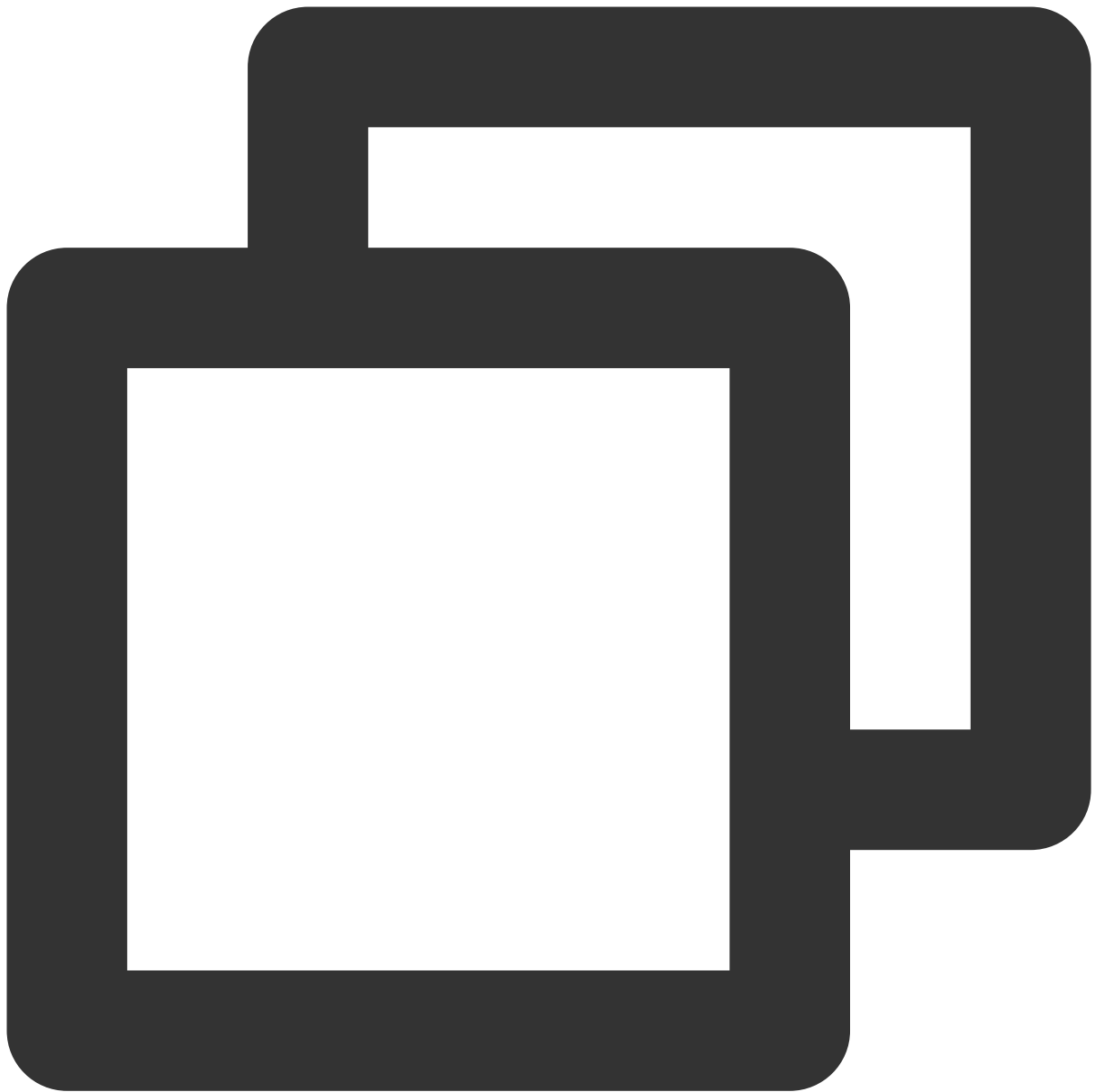
十一、监听小程序生命周期

可以通过**重写 BaseMiniAppProxy 的 onAppStateChange 方法**，实现监听小程序生命周期。

API说明：

参数 appState：当前小程序的生命周期状态，定义参考 AppState；

参数 MiniAppEvent：当前小程序的生命周期事件。



```
/**
 * 小程序生命周期事件回调
 * 调用上下文环境：主进程UI线程
 *
 * @param appState 事件状态
 * @param event 事件
 */
public abstract void onAppStateChange(@AppState int appState, MiniAppEvent event);
```


十二、定制基础库更新策略和监听

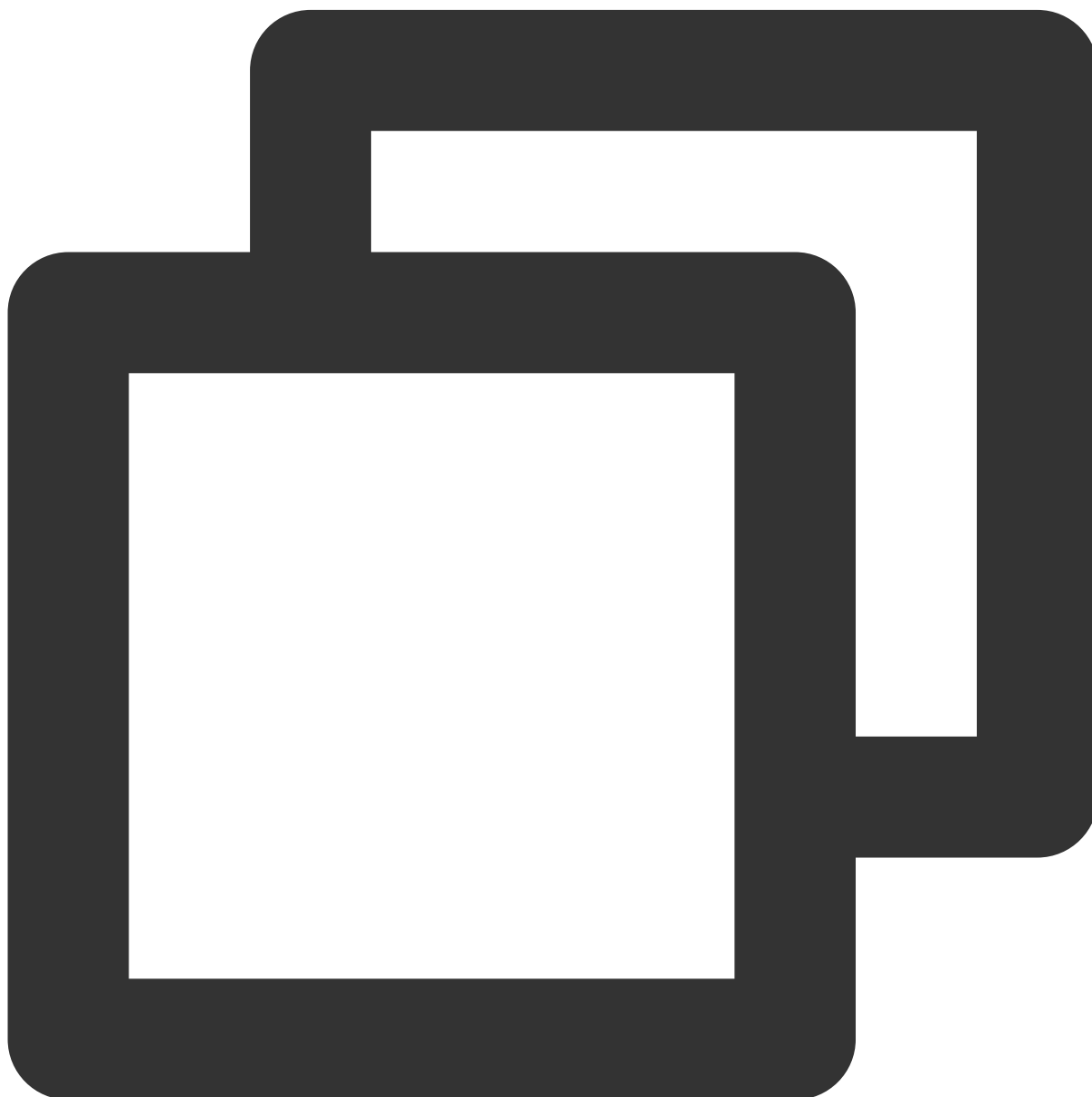
可以通过重写 **BaseMiniAppProxy** 的 **isUpdateBaseLib** 方法，实现监听小程序基础库的更新信息。

API说明：

参数 **context**：当前发起扫码的小程序进程上下文；

参数 **data**：小程序基础库的更新信息；

返回值：**true** 表示更新基础库，**false** 表示不更新基础库；默认值为 **false**。



```
/**
```

```
 * 基础库检测到更新时，是否进行更新
```

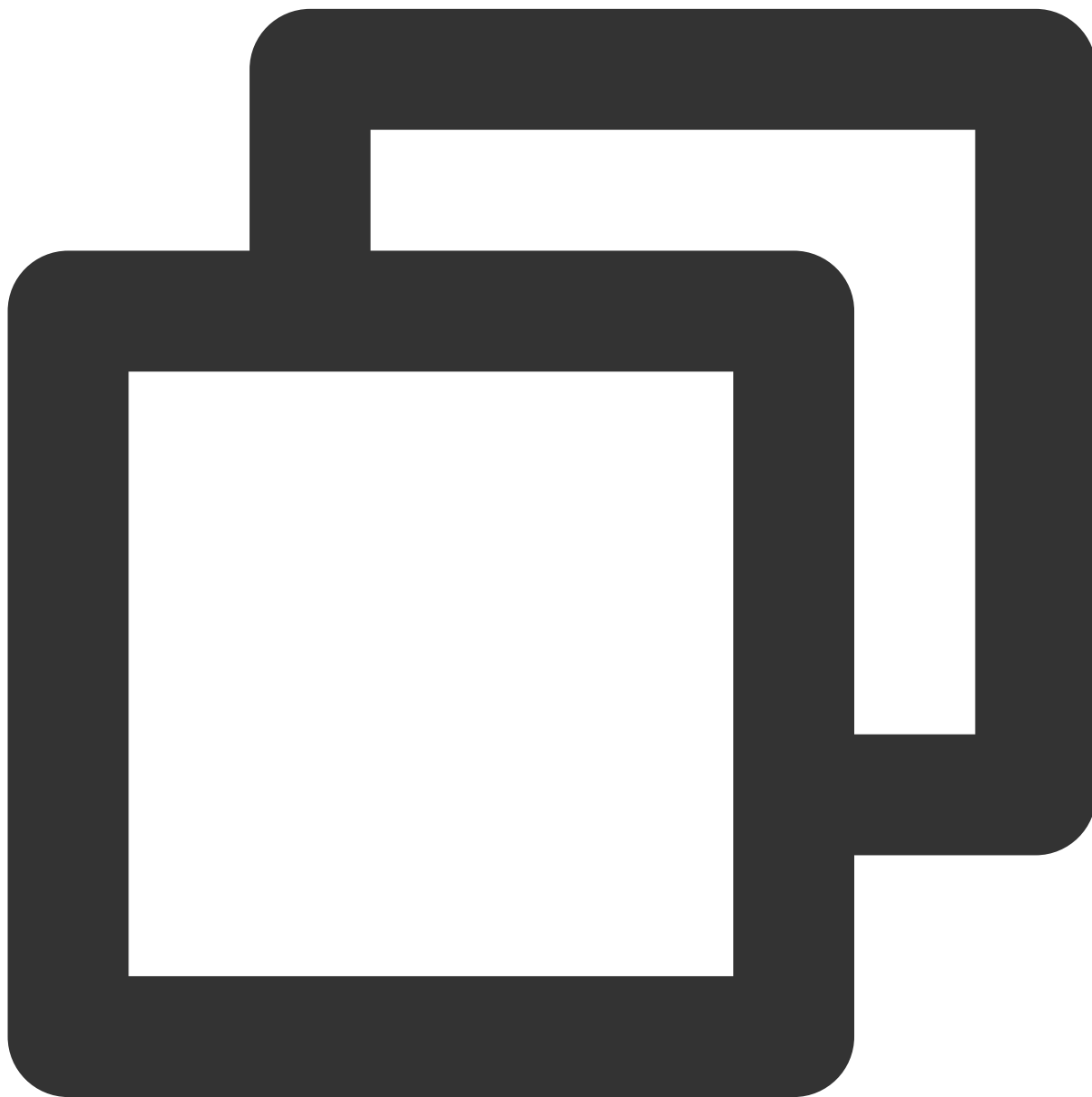
```
* @param context
* @param data 基础库信息数据
* @return true:更新;false:不更新,默认为 true
*/
public abstract boolean isUpdateBaseLib(Context context, JSONObject data);
```

定制分享能力

最近更新时间：2024-06-27 10:53:31

小程序 SDK 提供分享的接口桥接实现，用户可以参考如下方式定制分享能力：

1. 通过 MiniAppProxyImpl，在胶囊控制面板中添加自定义分享 Item。



```
private static final String SHARE_TWITTER = "twitter";
```

```
/**
```

```

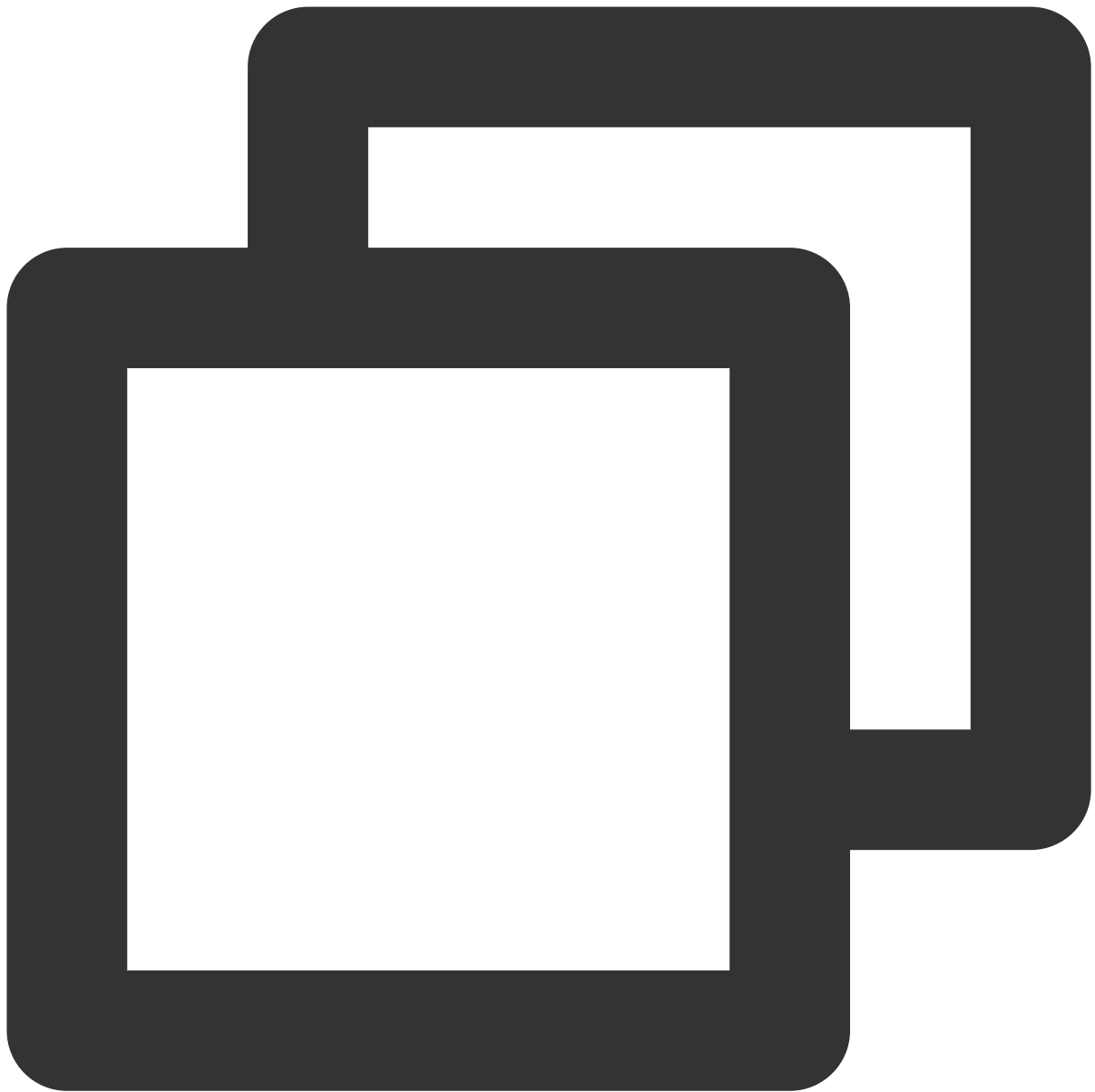
* 返回自定义分享数据Map
* 调用环境：子进程
*
* key:与getMoreItems方法中添加的MoreItem.id一致
* value:与getMoreItems方法中添加的MoreItem.shareKey一致
* @return
*/
@Override
public Map<String, Integer> getCustomShare(){
    Map<String, Integer> objects = new HashMap<>();
    objects.put(SHARE_TWITTER, ShareProxyImpl.OTHER_MORE_ITEM_2);
    return objects;
}

/**
* 返回胶囊更多面板的按钮，扩展按钮的ID需要设置为[100, 200]这个区间中的值，否则，添加无效
* 调用环境：子进程
*
* @param miniAppContext 小程序运行环境(小程序进程，非主进程)
* @param builder
* @return
*/
@Override
public ArrayList<MoreItem> getMoreItems(IMiniAppContext miniAppContext, MoreItemLis
    MoreItem item2 = new MoreItem();
    item2.id = ShareProxyImpl.OTHER_MORE_ITEM_2;
    item2.text = getString(miniAppContext,
        R.string.applet_mini_proxy_impl_other2);
    item2.shareKey = SHARE_TWITTER;//自定义分享的key,必须设置且唯一，与小程序端调用控制设置时
    item2.drawable = R.mipmap.mini_demo_about;

    builder.addMoreItem(item2)
    return builder.build();
}

```

2. 如需创建胶囊，可在**更多面板**中，单击**监听器**。



```
/**
 * 返回胶囊更多面板按钮单击监听器
 *
 * @return
 */
@Override
public OnMoreItemSelectedListener getMoreItemSelectedListener() {
    return new DemoMoreItemSelectedListener();
}

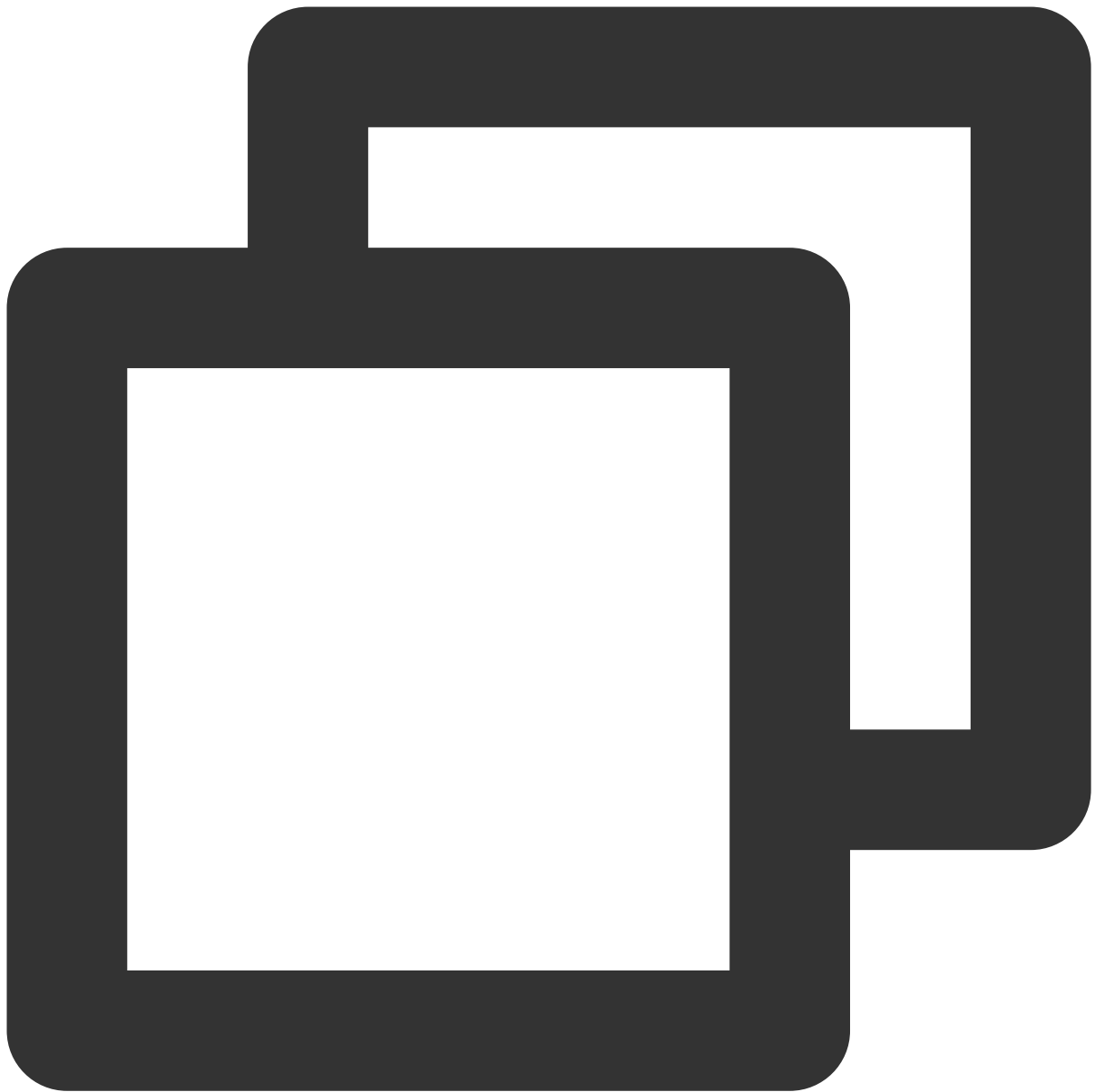
public class DemoMoreItemSelectedListener extends DefaultMoreItemSelectedListener {
```

```
public static final int CLOSE_MINI_APP = 150;

@Override
public void onMoreItemSelected(IMiniAppContext miniAppContext, int moreItemId)
    //处理开发者自定义点击事件(自定义分享事件除外)
    switch (moreItemId) {
        case CLOSE_MINI_APP:
            close(miniAppContext);
            return;
        case OTHER_MORE_ITEM_1:
            miniAppContext.getAttachedActivity().runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    Toast.makeText(miniAppContext.getAttachedActivity(), "custo
                }
            });
            return;
        }

        //处理内置分享和开发者自定义分享, 例如: 微博、twitter等
        super.onMoreItemSelected(miniAppContext, moreItemId);
    }
}
```

3. 按照如下类型接收分享, 单击**事件**, 开发人员在 `share` 方法中可以获取到分享数据, 并调用第三方 SDK 实现分享。



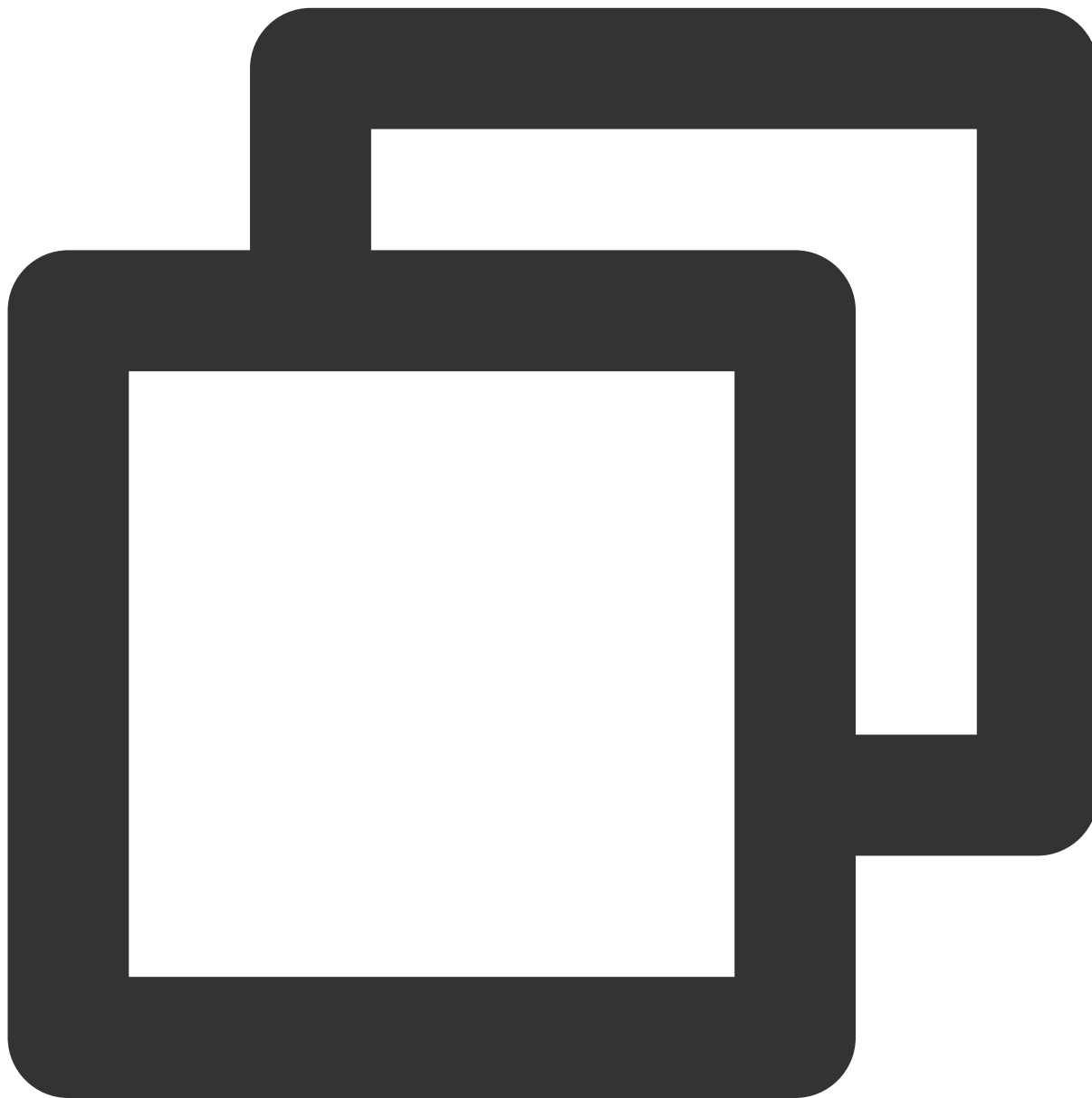
```
@ProxyService(proxy = ShareProxy.class)
public class ShareProxyImpl extends BaseShareProxy {
    /**
     * 分享
     *
     * @param shareData 分享数据
     */
    @Override
    public void share(Activity activity, ShareData shareData) {
        //todo share
    }
}
```

}

定制权限列表

最近更新时间：2024-06-27 10:53:20

调用 TmfMiniSDK 获取授权列表，示例如下：



```
/**
 * 获取小程序授权列表
 * @param appId
 * @param appVerType 小程序版本类型
 * @return
```

```
*/
public static List<MiniAuthState> getAuthStateList(String appId, int appVerType)

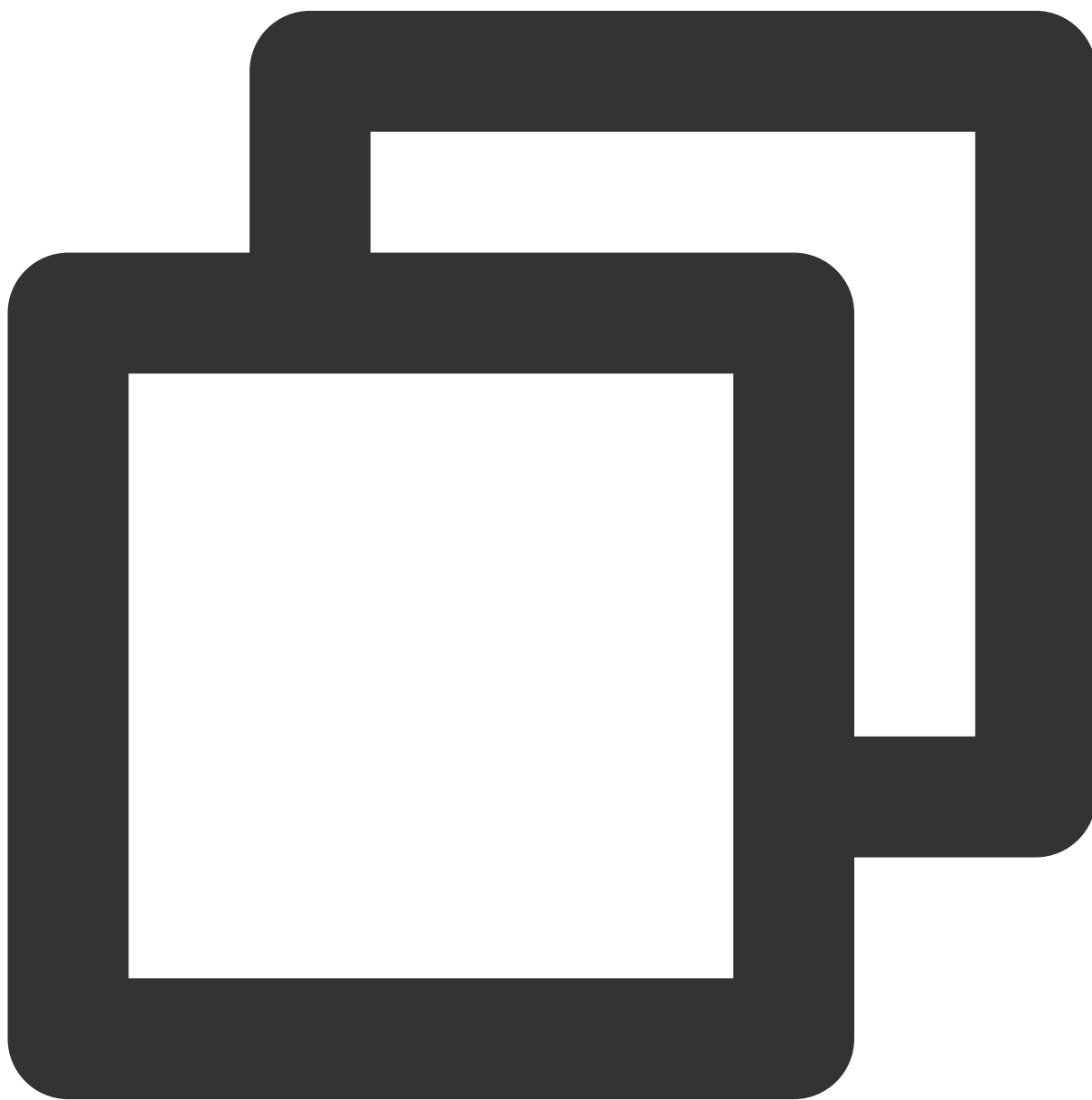
/**
 * 设置授权状态
 * @param appId
 * @param appVerType 小程序版本类型
 * @param scopeName 权限名
 * @param grant 是否授权
 */
public static void setAuthState(String appId, int appVerType, String scopeName, boo
```

定制权限申请

最近更新时间：2024-06-27 10:52:44

小程序申请系统权限

在使用小程序时，部分小程序 API 除需要申请小程序授权外，还需要申请对应的 Android 系统权限方可正常使用。SDK 默认会在使用这些 API 时自动弹窗向用户申请系统权限。您可以在初始化 SDK 时设置禁用自动权限申请，但是宿主 App 需要自行申请这些系统权限以保证小程序接口能够正常运行。



```
@ProxyService(proxy = MiniConfigProxy.class)
public class MiniConfigProxyImpl extends MiniConfigProxy {
    /**
     * 应用Application
     * @return
     */
    @Override
    public Application getApp() {
        return "app Application";
    }

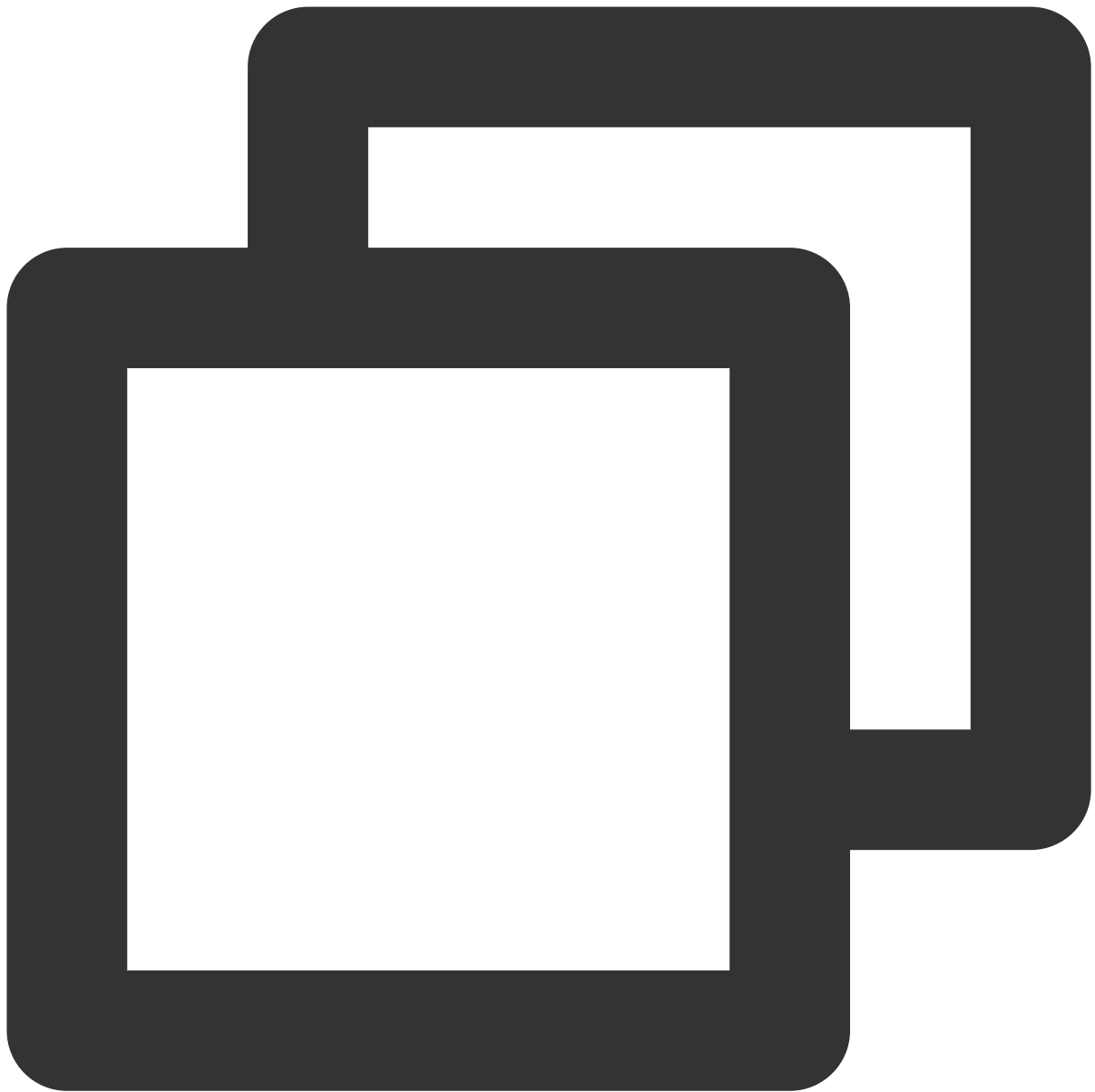
    /**
     * 创建初始化配置信息
     * @return
     */
    @Override
    public MiniInitConfig buildConfig() {
        MiniInitConfig.Builder builder = new MiniInitConfig.Builder();
        MiniInitConfig config = builder
            .configAssetName("tcmpp-android-configurations.json") //assets中配置
            .imei("IMEI") //配置设备id, 用于在管理平台上根据设备标识进行小程序的灰度发布
            .autoRequestPermission(false) //配置小程序使用到需要权限的API时是否自动向
            .debug(true) //日志开关, 默认关闭的
            .build();
    }
}
```

自定义系统权限申请

通过重写 `IPermissionManagerProxy` 接口，可以拦截小程序申请 Android 系统权限的事件并自定义权限申请逻辑。`IPermissionManagerProxy` 接口包含3个方法：`isPermissionGranted` 方法用于小程序检测宿主是否拥有某个系统权限；`requestForPermission` 和 `requestForPermissions` 方法用于小程序通知宿主申请某个或者某一组权限。

注意：

自定义系统权限申请后，初始化配置中的 `autoRequestPermission` 将不再生效。



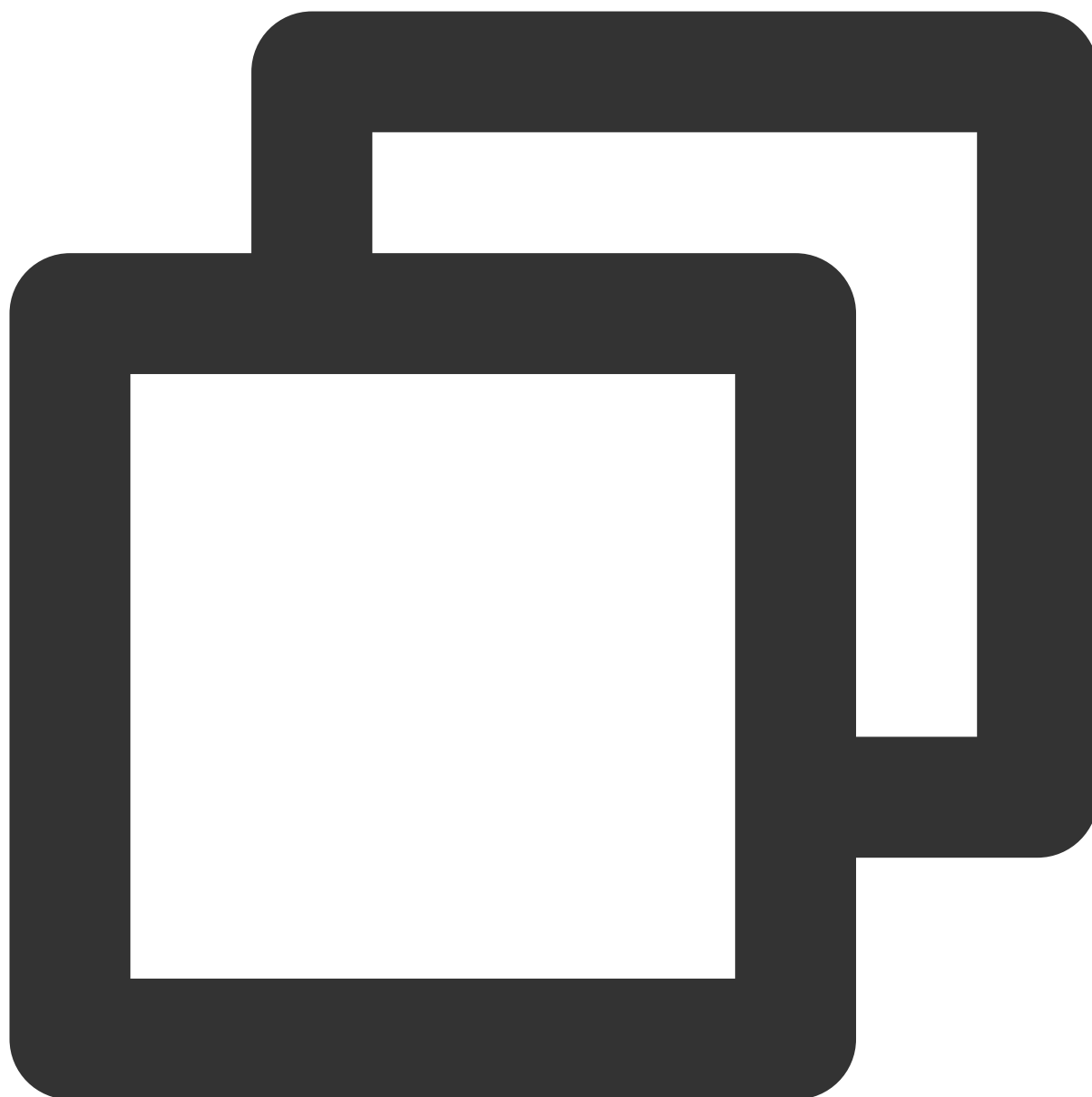
```
@ProxyService(proxy = IPermissionManagerProxy.class)
public class PermissionProxyImpl implements IPermissionManagerProxy {

    /**
     * 检查宿主是否拥有某个系统权限
     * @param context      Android 上下文
     * @param permission    待检查的系统权限, 参考 android.Manifest.permission 类
     * @return              宿主是否拥有权限
     */
    @Override
    public boolean isPermissionGranted(Context context, String permission) {
```

```
        return ContextCompat.checkSelfPermission(context, permission) == PackageManager.PERMISSION_GRANTED;
    }

    /**
     * 通知宿主请求系统权限
     * @param activity        申请权限的小程序 Activity
     * @param permission      待申请的系统权限，参考 android.Manifest.permission 类
     * @param callbacks       返回权限申请结果给小程序的回调
     */
    @Override
    public void requestPermission(Activity activity, String permission, RequestPermissionCallback callback) {
        Toast.makeText(activity, "正在申请 " + permission + " 权限", Toast.LENGTH_LONG).show();
        ActivityCompat.requestPermissions(activity, new String[] { permission }, 1);
    }

    /**
     * 通知宿主请求系统权限
     * @param activity        申请权限的小程序 Activity
     * @param permissions     待申请的系统权限列表，参考 android.Manifest.permission 类
     * @param callbacks       返回权限申请结果给小程序的回调
     */
    @Override
    public void requestPermissions(Activity activity, String[] permissions, RequestPermissionsCallback callback) {
        Toast.makeText(activity, "正在申请 " + permissions[0] + " 等权限", Toast.LENGTH_LONG).show();
        ActivityCompat.requestPermissions(activity, permissions, 1);
    }
}
```



```
/**
 * 通知系统权限申请结果
 */
interface RequestPermissionCallback {
    /**
     * 所有系统权限申请成功
     */
    void onSuccess();

    /**
     * 部分或者所有系统权限申请被拒绝
     */
}
```

```
* @param rejected      被拒绝的系统权限列表
*/
void onFail(String[] rejected);
}
```

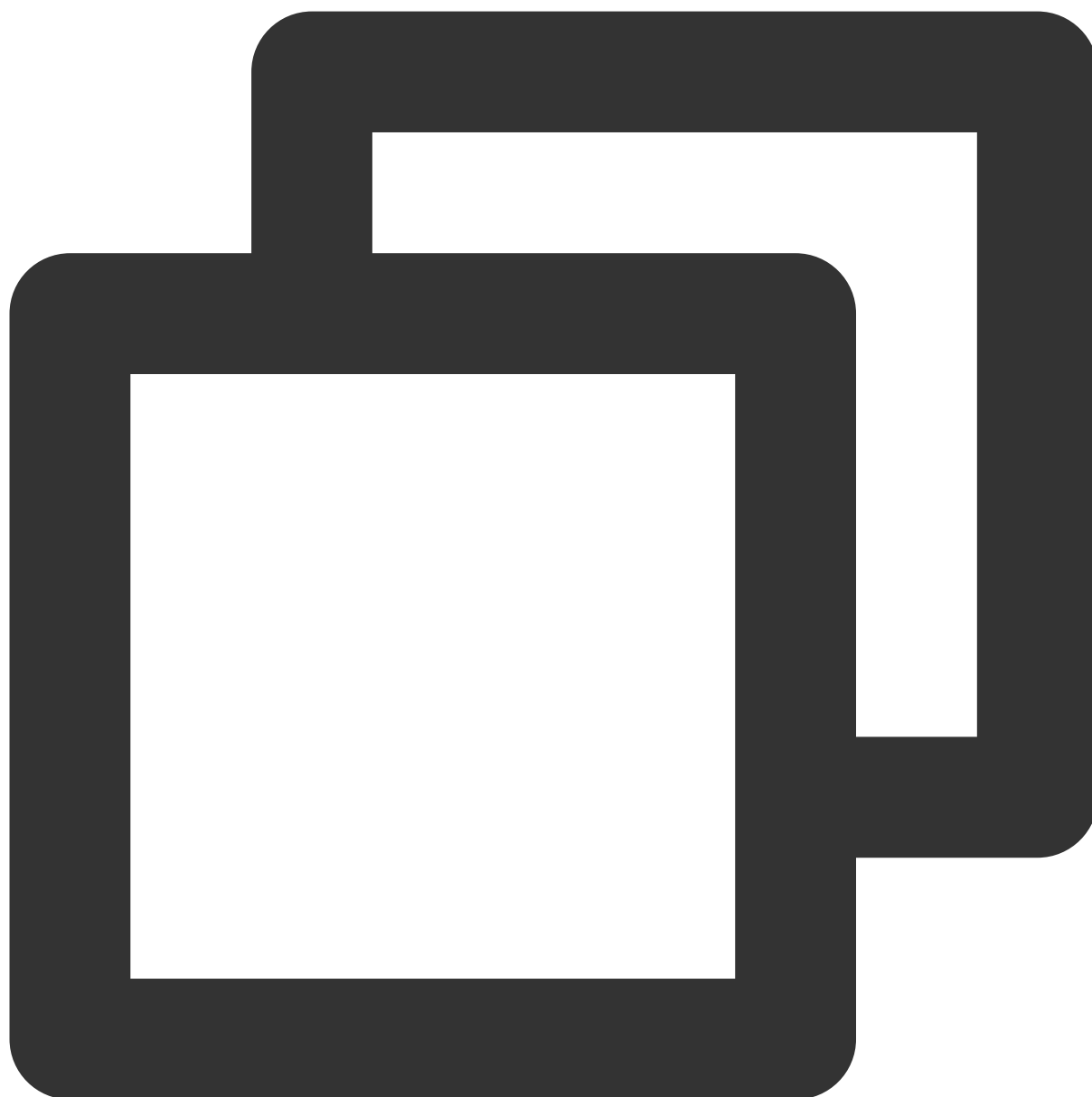

定制用户属性

最近更新时间：2024-06-27 10:52:34

宿主应用开发者可以通过 SDK 提供的方法设置用户的属性信息（地区或者账号），方便在灰度推送等场景下使用。

设置用户ID

API说明：参数 `userId` 表示设置账号信息。



```
/**
 * 设置账号信息
 * @param userId
 */
public static void setUserId(String userId)
```

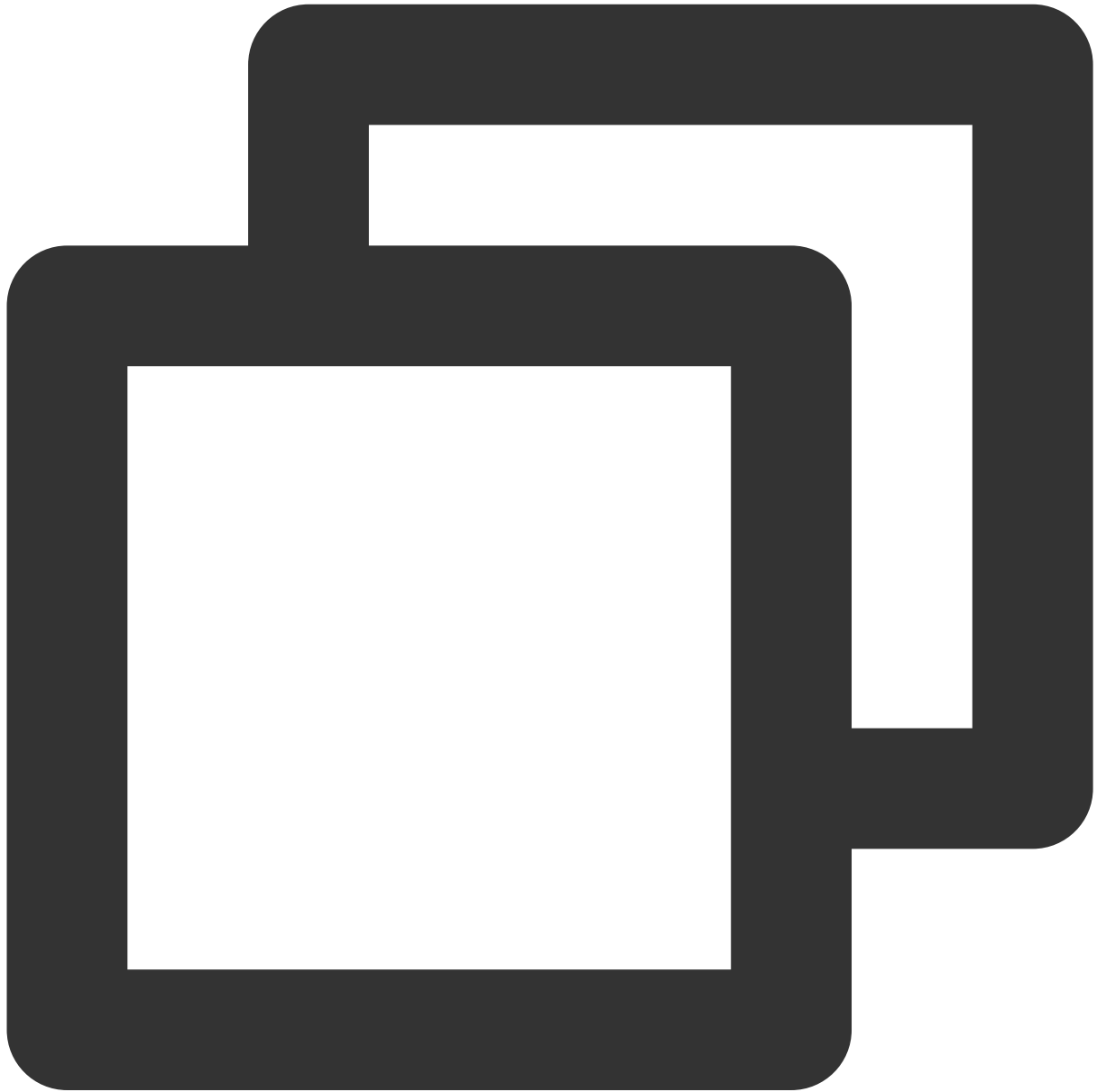
设置位置信息

API说明：

参数 country 表示用户所在国家。

参数 province 表示用户所在省。

参数 city 表示用户所在城市。



```
/**  
 * 设置位置信息  
 * @param country  
 * @param province  
 * @param city  
 */
```

```
public static void setLocation(String country, String province, String city)
```

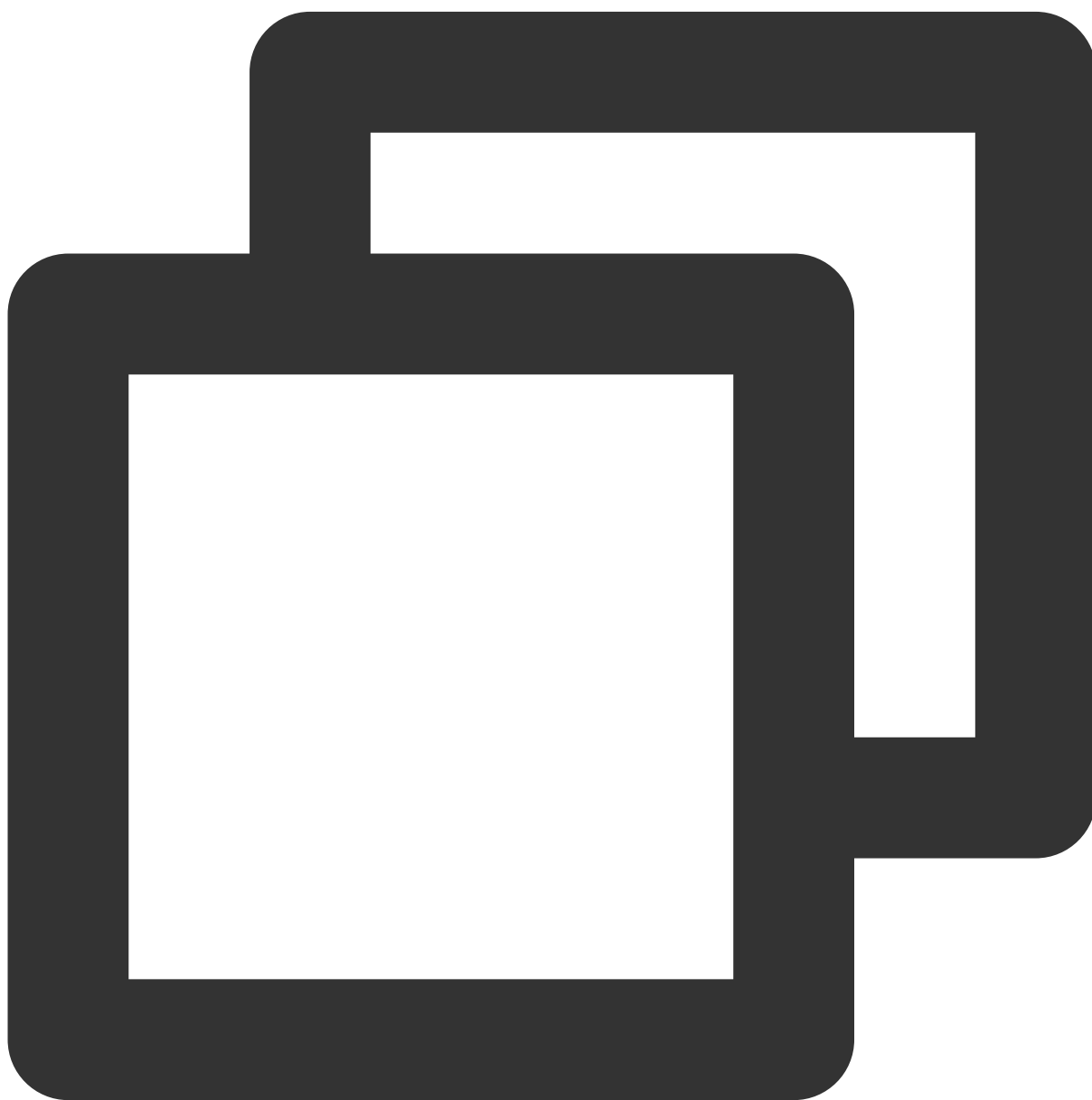
定制日志及事件上报

最近更新时间：2024-06-27 10:52:23

定制日志输出实现

通过实现 LogProxy 代理，可以实现小程序 SDK 内部日志的打印控制。

示例代码：



```
@ProxyService(proxy = LogProxy.class)
public class LogProxyImpl extends LogProxy {

    @Override
    public void log(int logLevel, String tag, String msg, Throwable t) {
        switch (logLevel) {
            case Log.DEBUG:
                if (t == null) {
                    android.util.Log.d(tag, msg);
                } else {
                    android.util.Log.d(tag, msg, t);
                }
                break;
            case Log.INFO:
                if (t == null) {
                    android.util.Log.i(tag, msg);
                } else {
                    android.util.Log.i(tag, msg, t);
                }
                break;
            case Log.WARN:
                if (t == null) {
                    android.util.Log.w(tag, msg);
                } else {
                    android.util.Log.w(tag, msg, t);
                }
                break;
            case Log.ERROR:
                if (t == null) {
                    android.util.Log.e(tag, msg);
                } else {
                    android.util.Log.e(tag, msg, t);
                }
                break;
            default:
                if (t == null) {
                    android.util.Log.v(tag, msg);
                } else {
                    android.util.Log.v(tag, msg, t);
                }
                break;
        }
    }

    @Override
    public boolean isColorLevel() {
```

```
        return true;
    }
}
```

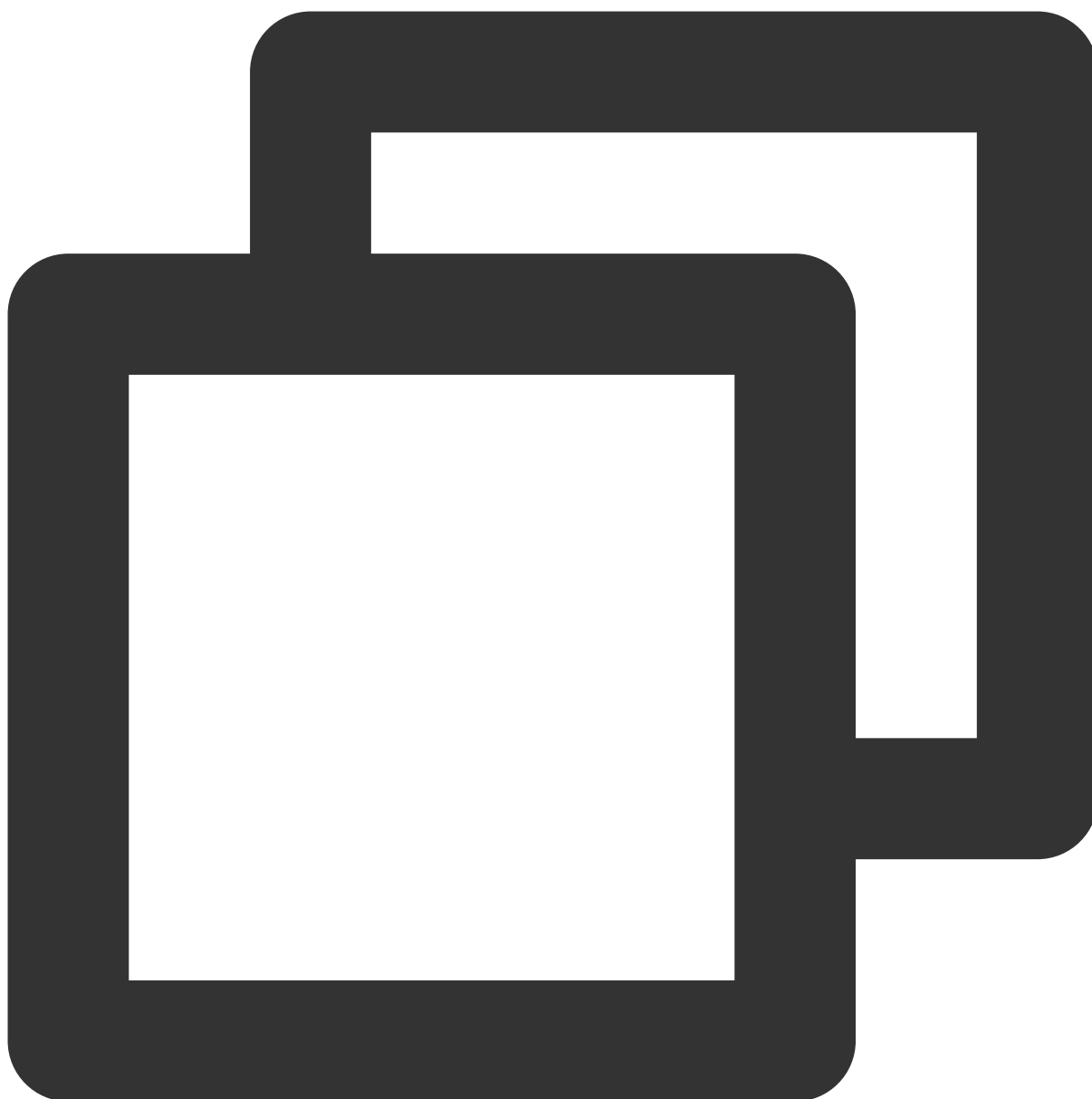
定制事件上报

宿主 APP 可以通过实现 **MiniAppProxy** 的 **reportMiniAppEvent** 方法定制小程序事件上报，从而覆盖 TCMP SDK 内部的上报逻辑。

说明：

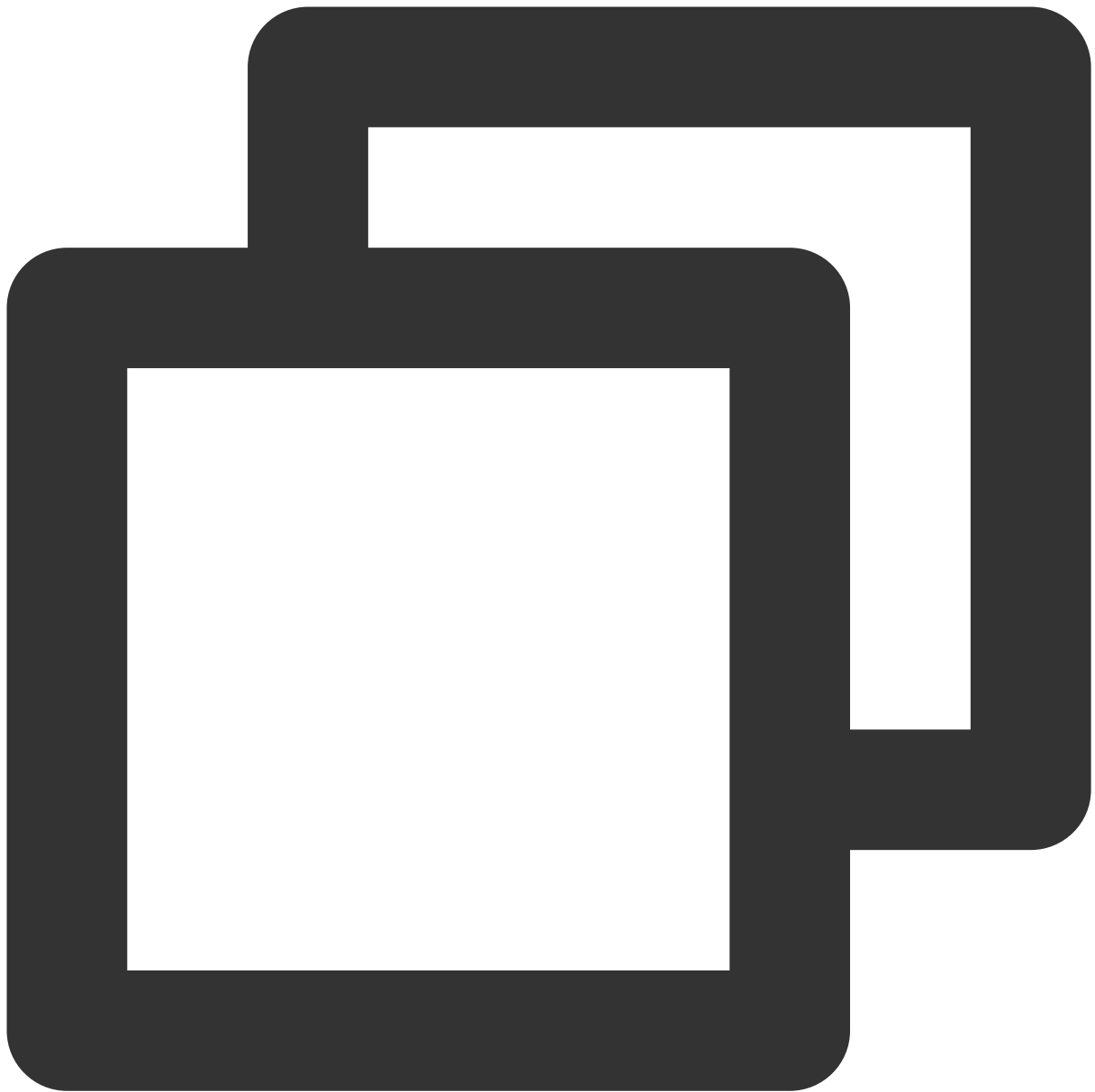
包括小程序操作事件及小程序内部调用 `wx.reportEvent` 上报的数据。

API 如下：



```
/**
 * 小程序事件上报代理
 * @param eId 事件ID
 * @param eventName 事件名字
 * @param props 事件属性
 * @param app 小程序信息
 * @return 为true时，SDK内部上报逻辑不再执行，使用用户自定义的事件上报逻辑
 */
public abstract boolean reportMiniAppEvent(int eId, String eventName, JSONObject pr
```

内置事件 ID 描述如下：



```
/** 自定义事件*/  
  
public static final int EID_None = 0;  
  
/** 打开小程序 */  
  
public static final int EID_OPEN_MINIAPP = 1;  
  
/** 更新小程序*/
```

```
public static final int EID_UPDATE_MINIAPP = 2;

/** 下载小程序 */

public static final int EID_DOWNLOAD_MINIAPP = 3;

/** 小程序页面浏览 */

public static final int EID_MINIAPP_PAGE_VIEW = 4;

/** 退出小程序 */

public static final int EID_EXIT_MINIAPP = 5;

/** 小程序行为事件*/

public static final int EID_MINIAPP_ACTION = 6;
```

定制日志上报

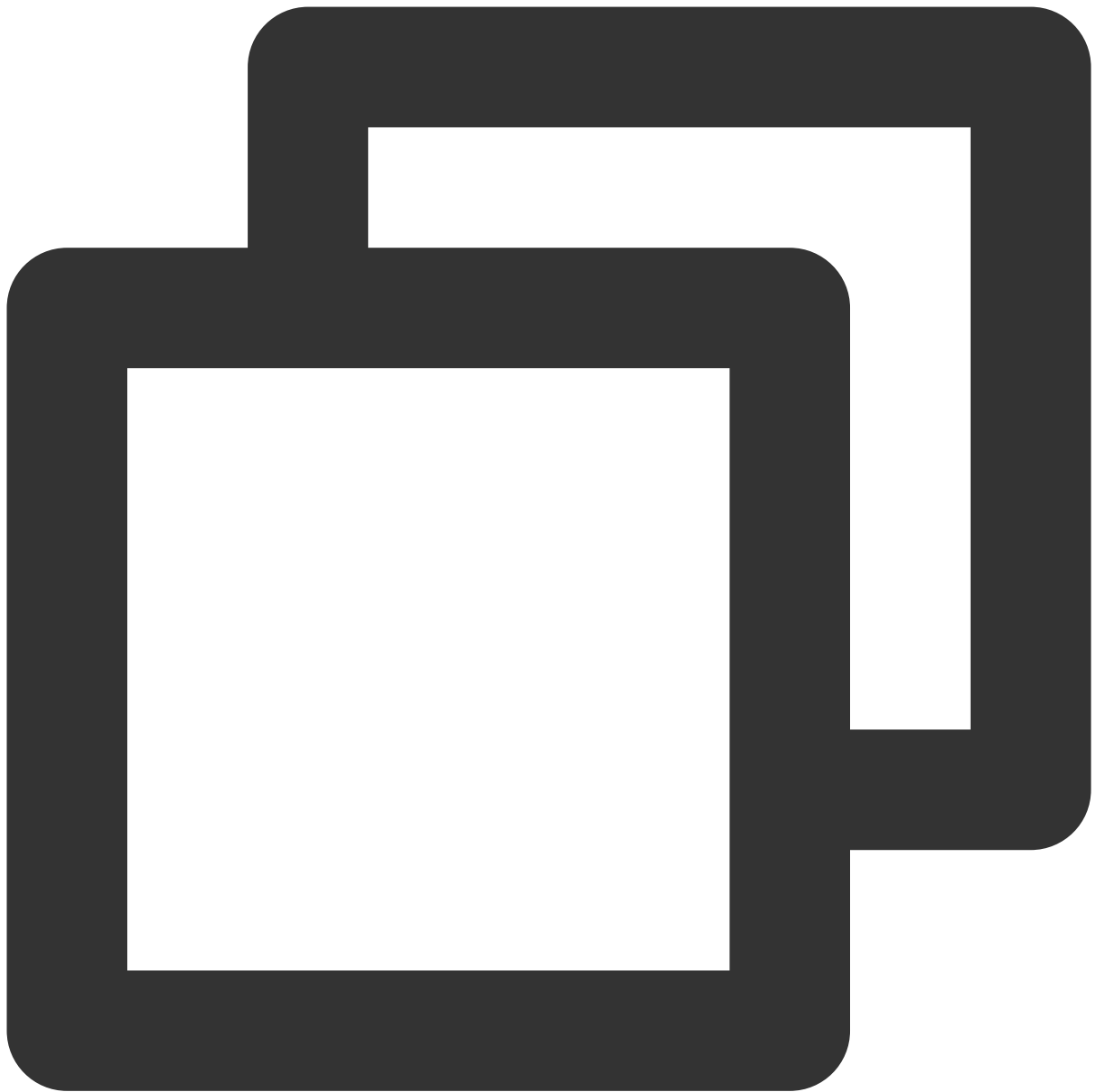
实时日志上报

宿主 APP 可以通过实现 **MiniAppProxy** 的 **reportRealTimeLog** 方法定制小程序实时事件日志上报逻辑。

说明：

包括小程序内部调用 `wx.getRealtimeLogManager` 写入的日志数据。

API 如下：



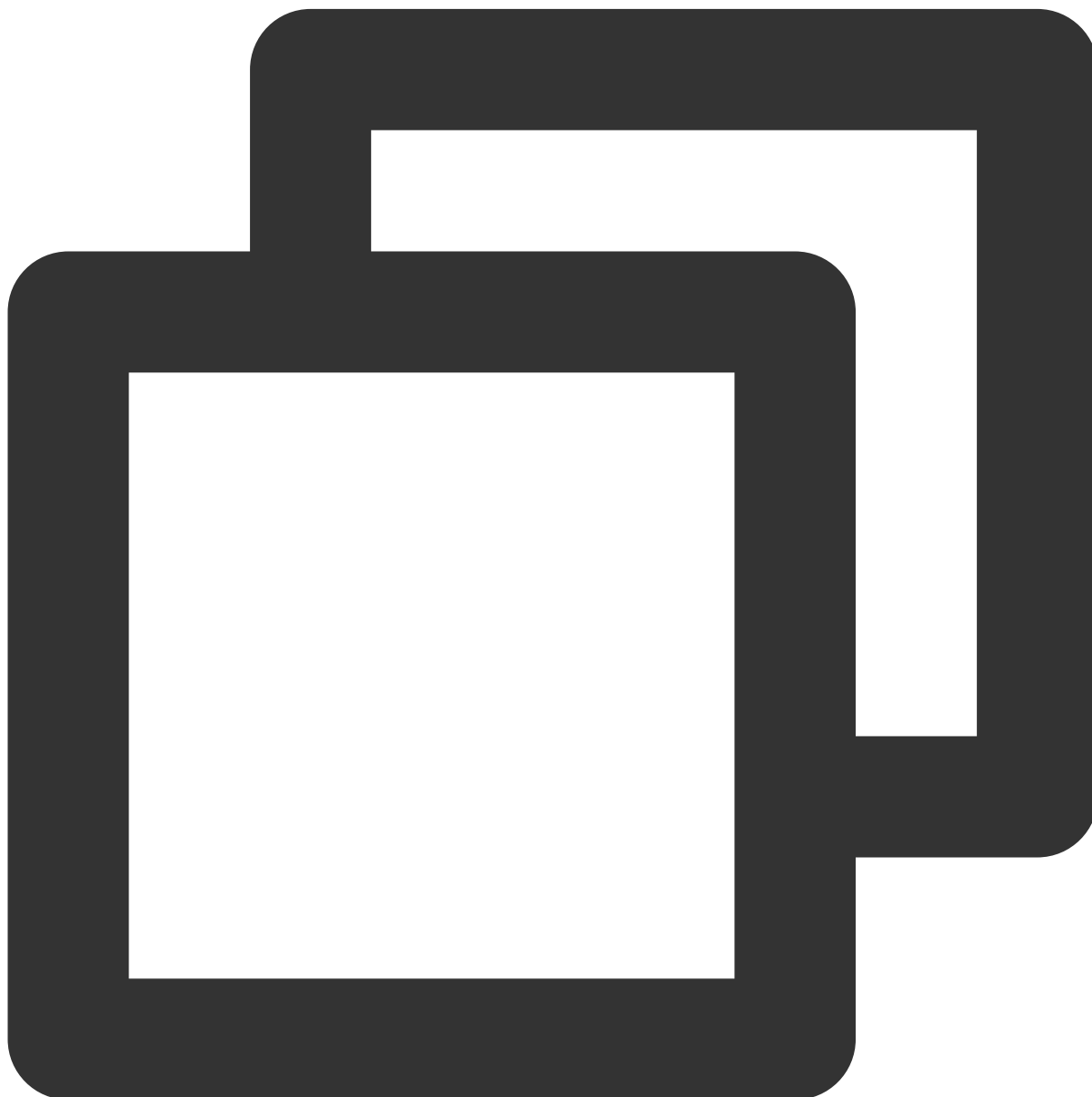
```
/**
 * 小程序实时日志上报代理
 * @param page 当前页面
 * @param jsLibVersion 基础库版本
 * @param app 小程序信息
 * @param filterMsgs 过滤关键字
 * @param logs 日志
 * @return 为true时，SDK内部上报逻辑不再执行
 */
public abstract boolean reportRealTimeLog(String page, String jsLibVersion, MiniApp
                                         ArrayList<RealTimeLogItem> logs);
```

小程序内部日志上报

宿主 APP 可以通过实现 **MiniAppProxy** 的 **uploadUserLog** 方法定制小程序的投诉反馈页面收集到日志上传逻辑。

说明：

包括小程序内部调用 `wx.getLogManager` 写入的日志数据，用户可以通过使用 `button` 组件的 `open-type="feedback"` 来上传打印的日志。



```
/**  
 * feedback 日志上传
```

```
*  
* @param appId 小程序appId  
* @param logPath 日志路径  
* @return 为true时，SDK内部上报逻辑不执行  
*/  
public abstract boolean uploadUserLog(String appId, String logPath);
```

开放接口

最近更新时间：2024-06-27 10:52:11

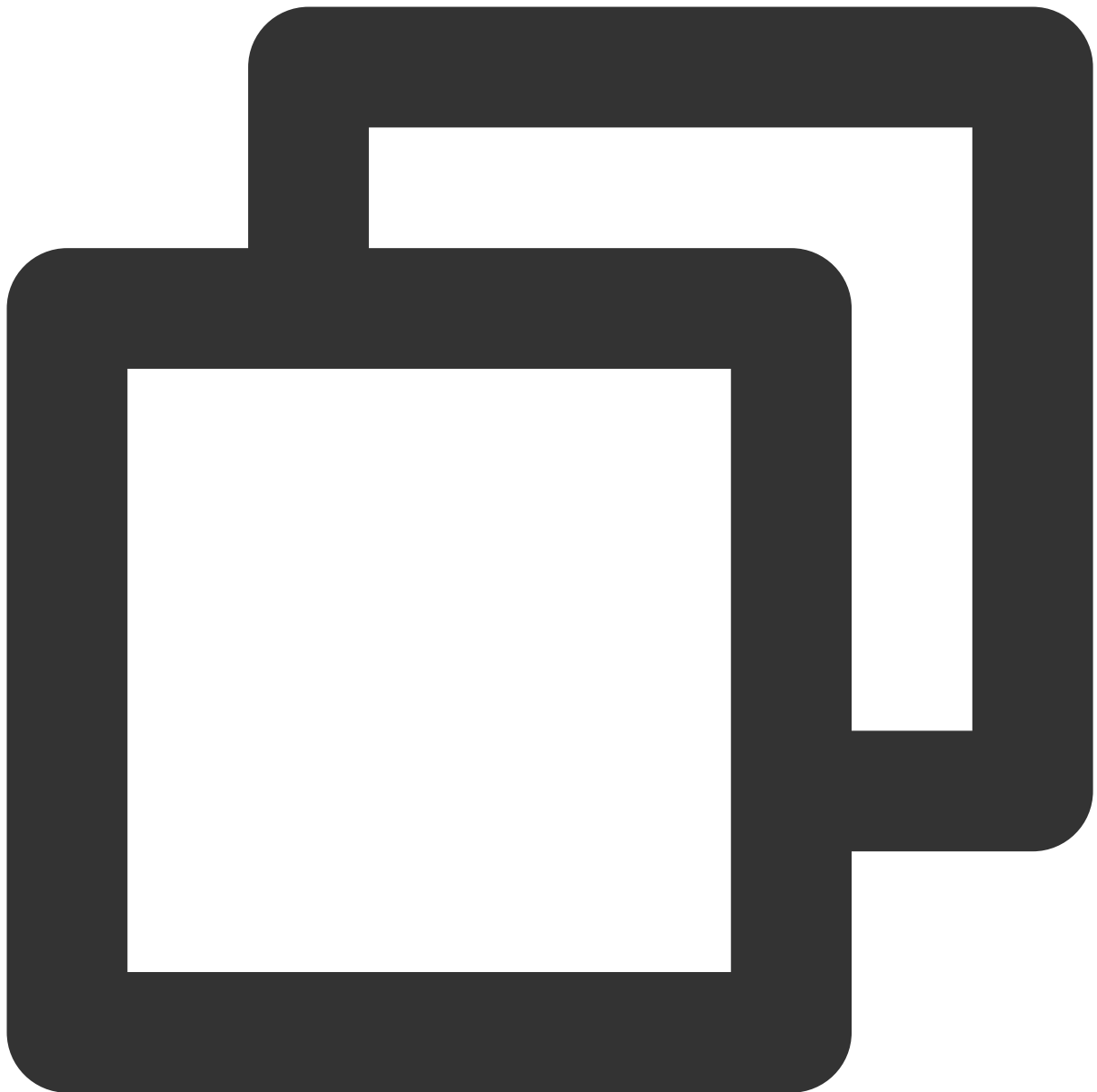
小程序 SDK 提供部分开放接口，用于调用和实现登录、获取用户信息、支付等宿主应用提供的能力；当前支持的开放接口如下表格所示：

小程序方法	MiniOpenApiProxy 方法	描述
wx.login	login	登录接口
wx.getUserInfo	getUserInfo	获取用户基本信息
wx.getUserProfile	getUserProfile	获取用户属性信息
wx.getPhoneNumber	getPhoneNumber	获取电话号码
wx.requestPayment	requestPayment	发起支付
wx.checkSession	checkSession	检查登录状态是否过期

用户可以通过实现 MiniOpenApiProxy 代理，将小程序和宿主应用的数据交互进行关联，示例代码如下：

说明：

- 参数 IMiniAppContext 为当前小程序的上下文信息；
- 参数 JSONObject 为小程序调用开放接口时传递的参数；
- 参数 AsyncResult 为异步回调接口，用户需要通过此接口将宿主应用实现开放接口的返回值返回给小程序。



```
@ProxyService(proxy = MiniOpenApiProxy.class)
public class MiniOpenApiProxyImpl implements MiniOpenApiProxy {
    private static final String TAG = "MiniOpenApiProxyImpl";
    @Override
    public void login(IMiniAppContext miniAppContext, JSONObject params, AsyncResul
        QMLog.d(TAG, "login:" + params);
        JSONObject jsonObject = new JSONObject();
        try {
            jsonObject.put("key", "wx.login");
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }
}
```

```
    }
    result.onReceiveResult(true, jsonObject);
}

@Override
public void checkSession(IMiniAppContext miniAppContext, JSONObject params, AsyncTask.OnReceiveResultListener listener) {
    QMLog.d(TAG, "checkSession:" + params);
    JSONObject jsonObject = new JSONObject();
    try {
        jsonObject.put("key", "wx.checkSession");
    } catch (JSONException e) {
        e.printStackTrace();
    }
    result.onReceiveResult(true, jsonObject);
}

@Override
public void getUserInfo(IMiniAppContext miniAppContext, JSONObject params, AsyncTask.OnReceiveResultListener listener) {
    QMLog.d(TAG, "getUserInfo:" + params);
    JSONObject jsonObject = new JSONObject();
    try {
        final JSONObject userInfo = new JSONObject();

        userInfo.put("nickName", "userInfo测试");
        // userInfo.put("avatarUrl", bundle.getString("avatarUrl"));

        userInfo.put("gender", 0);
        userInfo.put("country", "CN");
        userInfo.put("province", "BeiJing");
        userInfo.put("city", "BeiJing");
        userInfo.put("language", "en");
        jsonObject.put("userInfo", userInfo);
    } catch (JSONException e) {
        e.printStackTrace();
    }
    result.onReceiveResult(true, jsonObject);
}

@Override
public void getUserProfile(IMiniAppContext miniAppContext, JSONObject params, AsyncTask.OnReceiveResultListener listener) {
    QMLog.d(TAG, "getUserProfile:" + params);
    JSONObject jsonObject = new JSONObject();
    try {
        jsonObject.put("key", "wx.getUserProfile");
    } catch (JSONException e) {
        e.printStackTrace();
    }
}
```



```
        result.onReceiveResult(true, jsonObject);
    }

    @Override
    public void getPhoneNumber(IMiniAppContext miniAppContext, JSONObject params, A
        QMLog.d(TAG, "getPhoneNumber:" + params);
        JSONObject jsonObject = new JSONObject();
        try {
            jsonObject.put("key", "wx.getPhoneNumber");
        } catch (JSONException e) {
            e.printStackTrace();
        }
        result.onReceiveResult(true, jsonObject);
    }

    @Override
    public void requestPayment(IMiniAppContext miniAppContext, JSONObject params, A
        QMLog.d(TAG, "requestPayment:" + params);
        JSONObject jsonObject = new JSONObject();
        try {
            jsonObject.put("key", "wx.requestPayment");
        } catch (JSONException e) {
            e.printStackTrace();
        }
        result.onReceiveResult(true, jsonObject);
    }
```

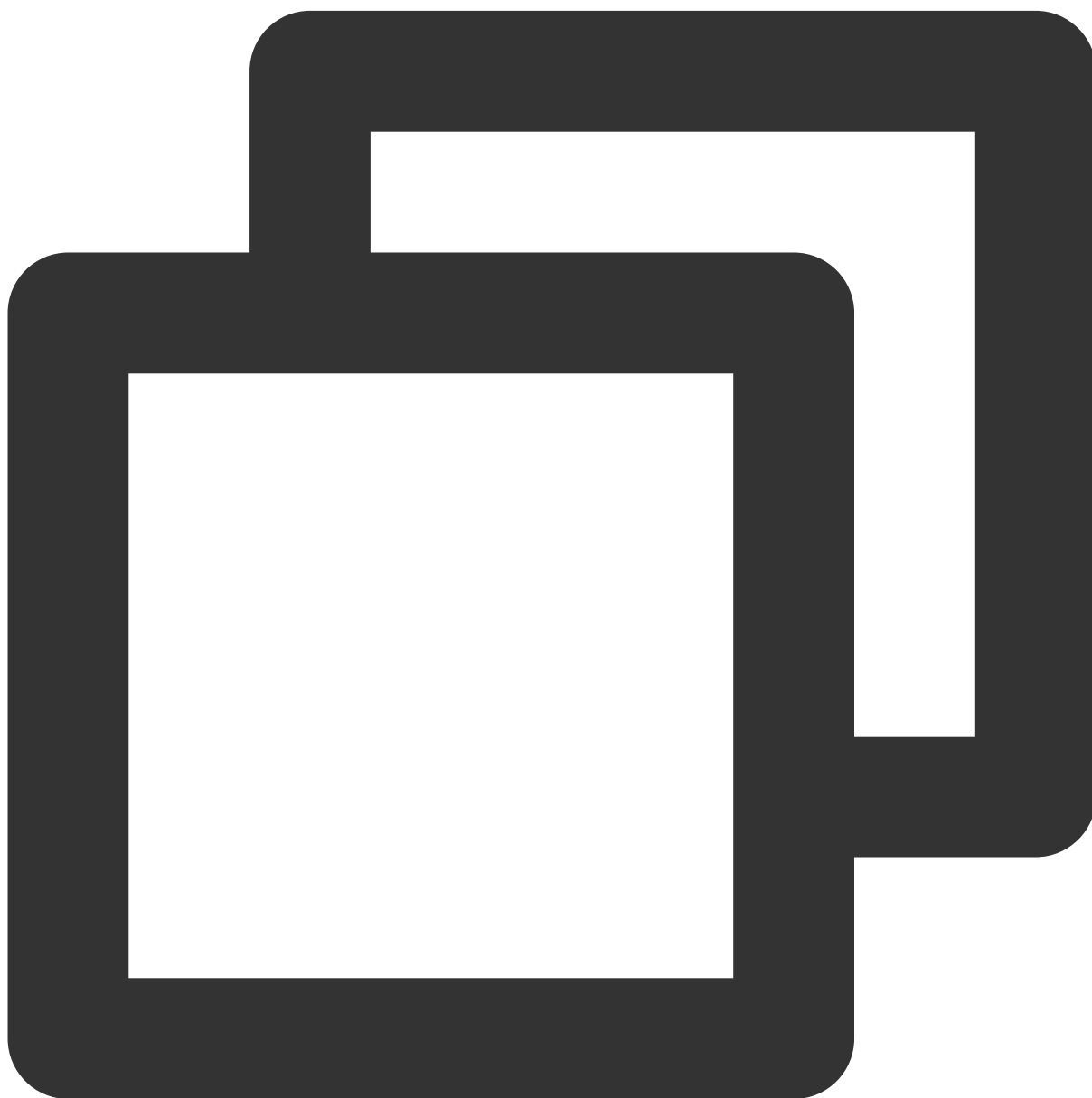
其他

最近更新时间：2024-08-13 17:10:36

WebView 组件中特殊链接处理

如果 **WebView** 组件展示的网页中存在特殊的 URL 链接，例如 `tcmpp://host/path` 这种以自定义 **scheme** 开头的链接，宿主可以接管这些链接的跳转操作，并自定义跳转行为。

接管链接跳转应当创建类实现 **IWebViewLinkProxy** 接口，并添加 **ProxyService** 注解：



```
@ProxyService(proxy = IWebViewLinkProxy.class)
public class CustomWebViewLinkProxy implements IWebViewLinkProxy {

    /**
     * Handle custom url loading in web-view component.
     * Custom url is url with any custom scheme (scheme not network protocol such a
     * @param miniContext context of mini-program
     * @param url the url which is loading
     * @return return true if the url loading is handled, otherwise false the web-v
     */
    @Override
    public boolean webViewCustomUrlLoading(IMiniAppContext miniContext, String url) {
        Uri uri = Uri.parse(url);
        if ("tcmpp".equals(uri.getScheme())) {
            Intent intent = new Intent(Intent.ACTION_VIEW, uri);
            miniContext.getAttachedActivity().startActivity(intent);
            return true;
        }
        return false;
    }
}
```

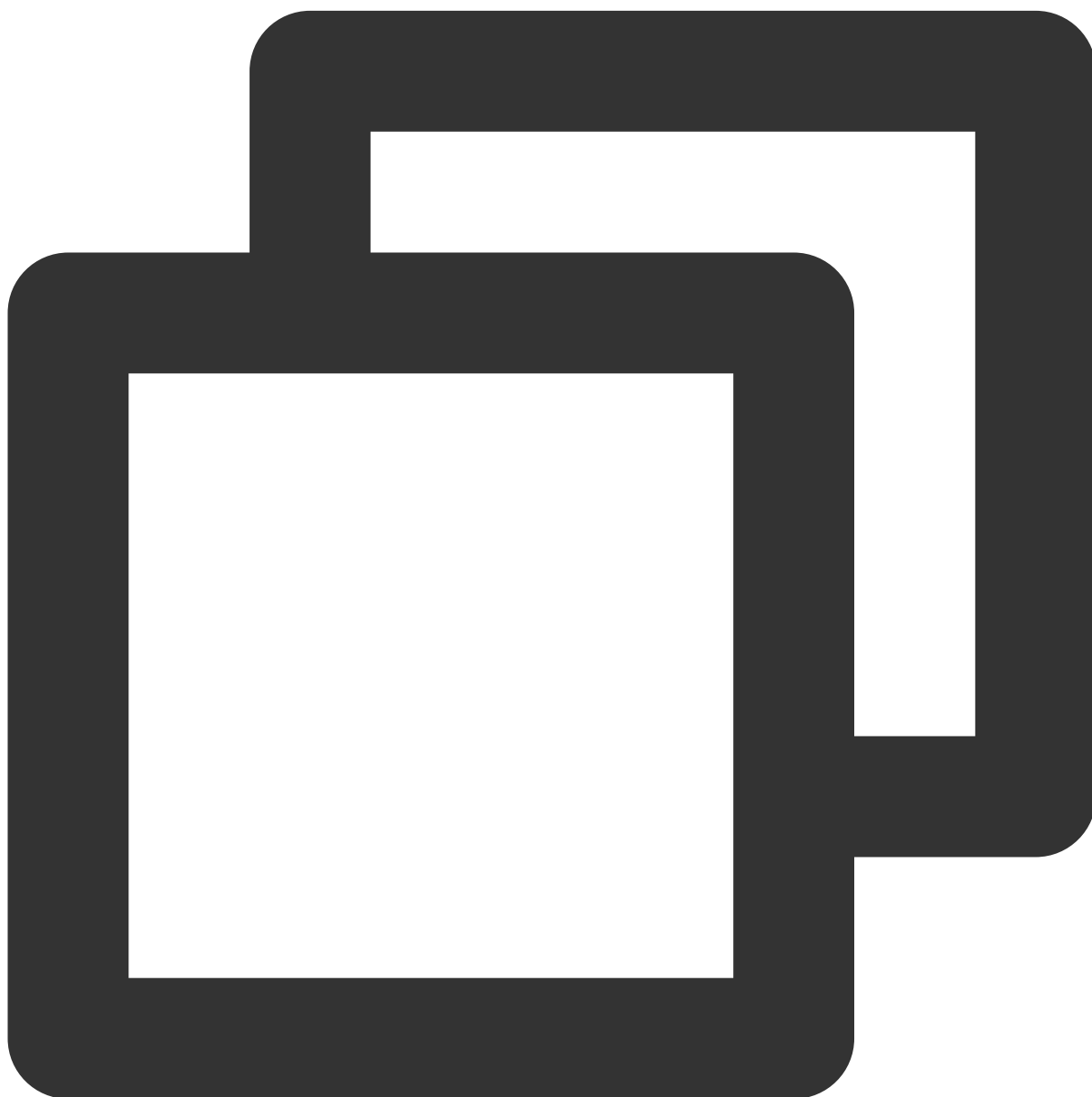
实现方法中提供了小程序上下文以及被加载的链接。如果开发者处理了链接跳转并返回 **true**，**web-view** 将不再处理该链接。如果返回 **false**，**web-view** 将按照正常逻辑处理该链接。

API 说明

最近更新时间：2024-07-29 16:56:21

MiniStartOptions

小程序启动参数配置



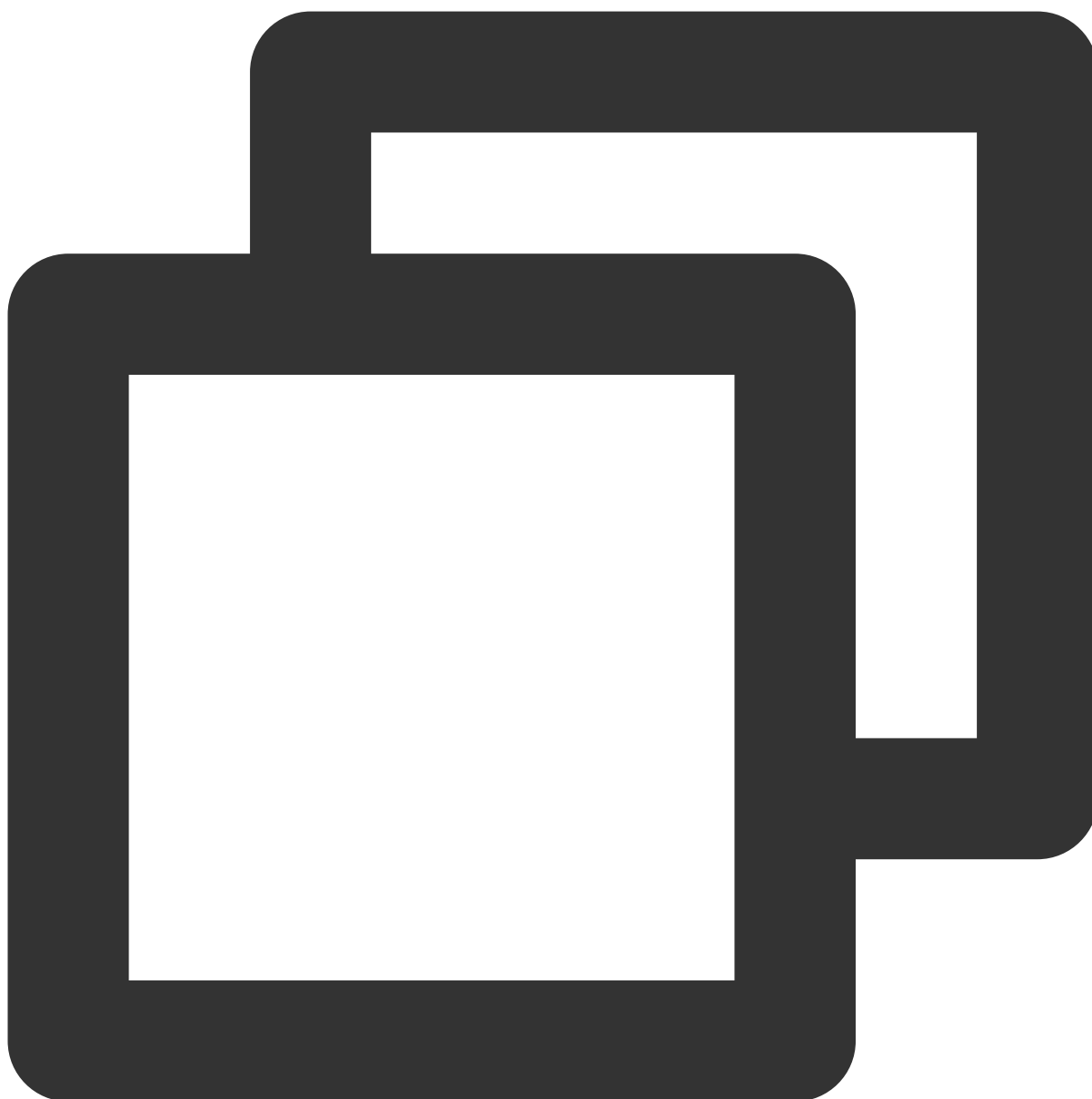
```
/**
 * 打开小程序时是否强制检查更新 (APP每次启动第一次打开小程序有效), false: 优先使用本地缓存, 同时异步
 */
public boolean isForceUpdate = false;
/**
 * 入口地址, 支持添加参数: path?key=value&key1=value1
 */
public String entryPath;
/**
 * 接收小程序启动过程中错误信息
 */
public ResultReceiver resultReceiver;

/**
 * 小程序启动参数
 */
public String params;

/**
 * 设置打开小程序时使用的任务模式
 *
 * false: 多任务模式, true: 为单任务模式
 */
public boolean isSingleTask;
```

MiniCode

返回错误码描述。



```
/**
 * 成功
 */
public static final int CODE_OK = 0;

////////////////////服务端错误////////////////////////////////////
public static final int C_SERVER = -11000;
/**
 * shark网络错误
 */
public static final int C_SERVER_SHARK_ERROR = -11001;
```

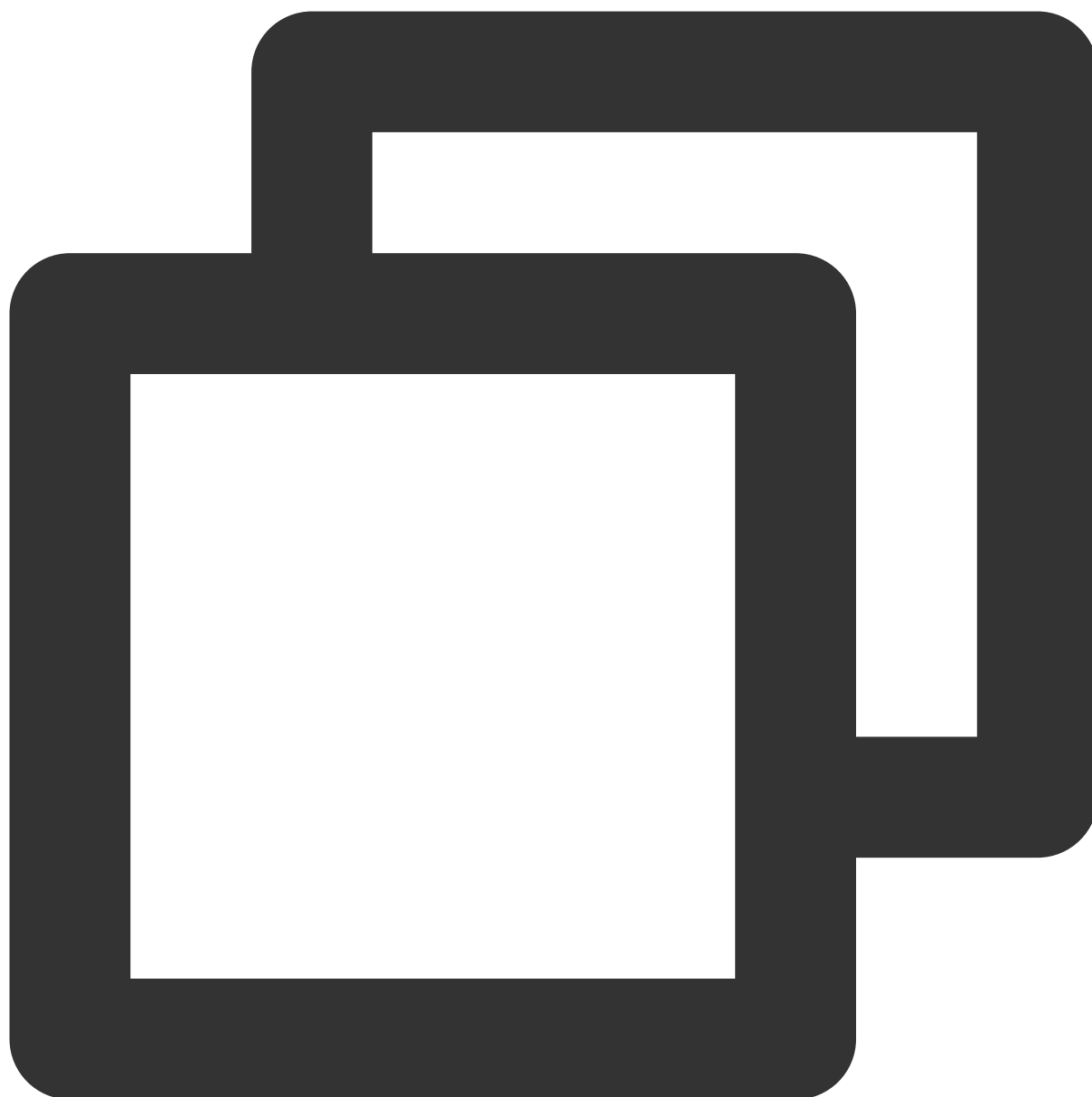
```
/**
 * 服务端返回code错误
 */
public static final int C_SERVER_RET_CODE_ERROR = -11002;
/**
 * 服务端返回response为空
 */
public static final int C_SERVER_RESPONSE_NULL = -11003;
/**
 * 服务端返回小程序更新类型有问题
 */
public static final int C_SERVER_UPDATE_TYPE_ERROR = -11004;
/**
 * 服务端返回数据解析异常
 */
public static final int C_SERVER_PARSE_DATA_ERROR = -11005;
/**
 * 小程序不存在或已下架
 */
public static final int C_SERVER_TAKE_OFF = -11006;
/**
 * 达到月活上限
 */
public static final int C_SERVER_MAU_LIMIT = -11007;
/**
 * 资源限额
 */
public static final int C_SERVER_RES_LIMIT = -11008;
/**
 * 单个接口请求频率过高
 */
public static final int C_SERVER_FREQ_LIMIT_API = -11011;

////////////////////////////////////客户端错误////////////////////////////////////
public static final int C_CLIENT = -12000;
/**
 * shark实例空
 */
public static final int C_CLIENT_SHARK_IS_NULL = -12001;
/**
 * 预览小程序需要先登录
 */
public static final int C_CLIENT_NEED_LOGIN_PREVIEW_APP = -12002;
/**
 * 数据解析异常
 */
public static final int C_CLIENT_JSON_EXCEPTION = -12003;
```

```
/**
 * 扫码异常
 */
public static final int C_CLIENT_SCAN_ERROR = -12004;
/**
 * 小程序信息缺失
 */
public static final int C_CLIENT_MINI_APP_INFO_ERROR = -12005;
/**
 * 扫码错误
 */
public static final int C_CLIENT_QRCODE_ERROR = -12006;
/**
 * 非tmf小程序二维码
 */
public static final int C_CLIENT_QRCODE_INVALIDATE = -12007;
/**
 * appId空
 */
public static final int C_CLIENT_APPID_EMPTY = -12008;
/**
 * businessId null
 */
public static final int C_CLIENT_QRCODE_BUSINESSID_NULL = -12009;
/**
 * 小程序启动异常
 */
public static final int C_CLIENT_START_MINI_APP_THROWABLE = -12010;
/**
 * json解析异常
 */
public static final int C_CLIENT_JSON_ERROR = -12011;
/**
 * 小程序下载失败
 */
public static final int C_CLIENT_MINI_APP_DOWNLOAD_FAIL = -12012;
/**
 * 小程序解析失败
 */
public static final int C_CLIENT_MINI_APP_PARSE_FAIL = -12013;
```

MiniApp

小程序信息描述类。

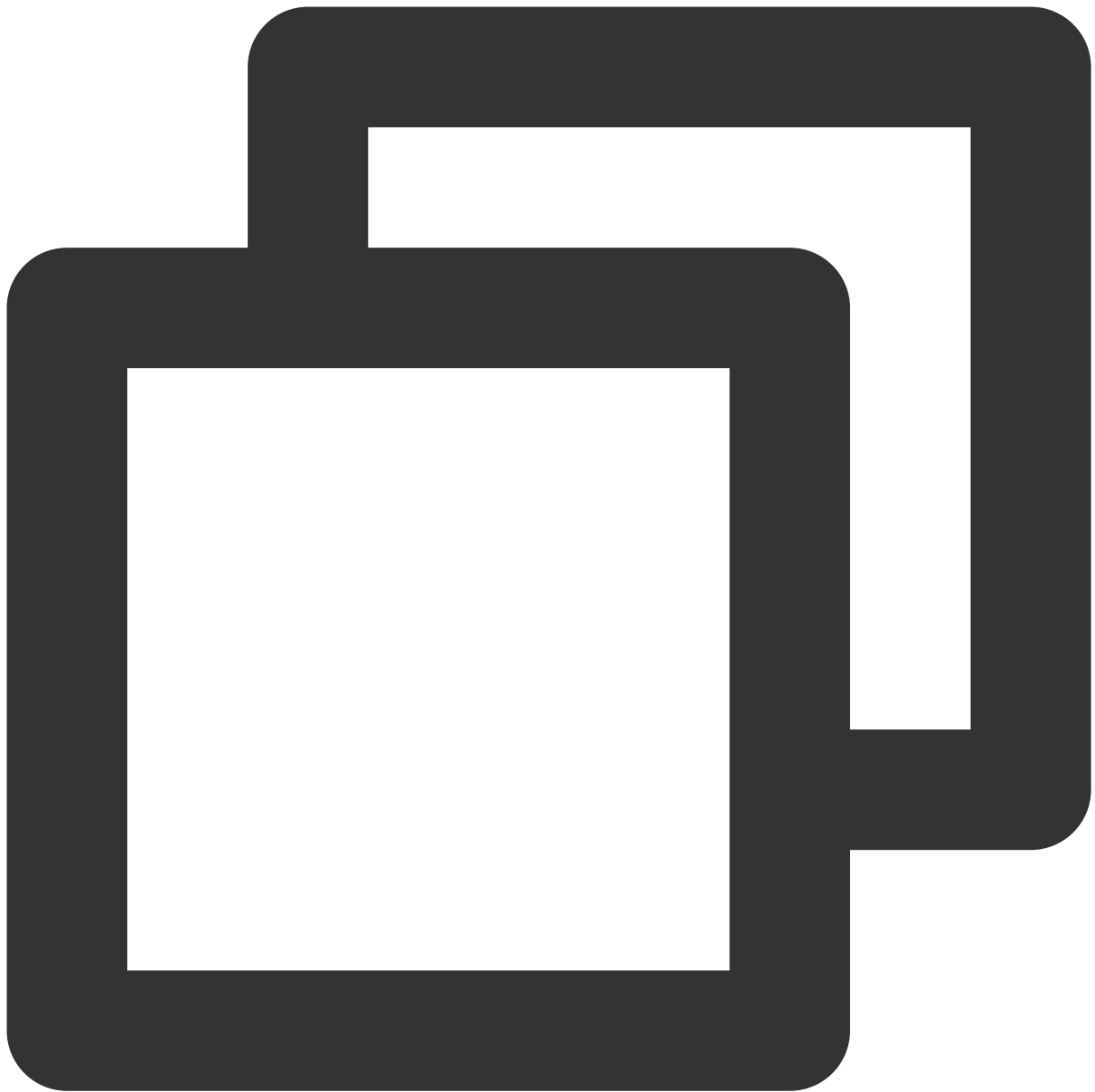


```
/**
 * 正式小程序
 */
public static final int TYPE_ONLINE = MiniSDKConst.ONLINE;
/**
 * 调试小程序
 */
public static final int TYPE_DEVELOP = MiniSDKConst.DEVELOP;
/**
 * 预览小程序
```

```
    */
    public static final int TYPE_PREVIEW = MiniSDKConst.PREVIEW;
    /**
     * 体验小程序
     */
    public static final int TYPE_EXPERIENCE = MiniSDKConst.EXPERIENCE;
    /**
     * 小程序id
     */
    public String appId;
    /**
     * 小程序版本类型(正式、预览、开发版)
     */
    public int appVerType;
    /**
     * 小程序版本
     */
    public String version;
    /**
     * 小程序名
     */
    public String name;
    /**
     * 小程序图标
     */
    public String iconUrl;
    /**
     * 小程序简介
     */
    public String appIntro;
    /**
     * 开发者企业名称
     */
    public String appDeveloper;
    /**
     * 时间戳
     */
    public long time;
```

MiniScene

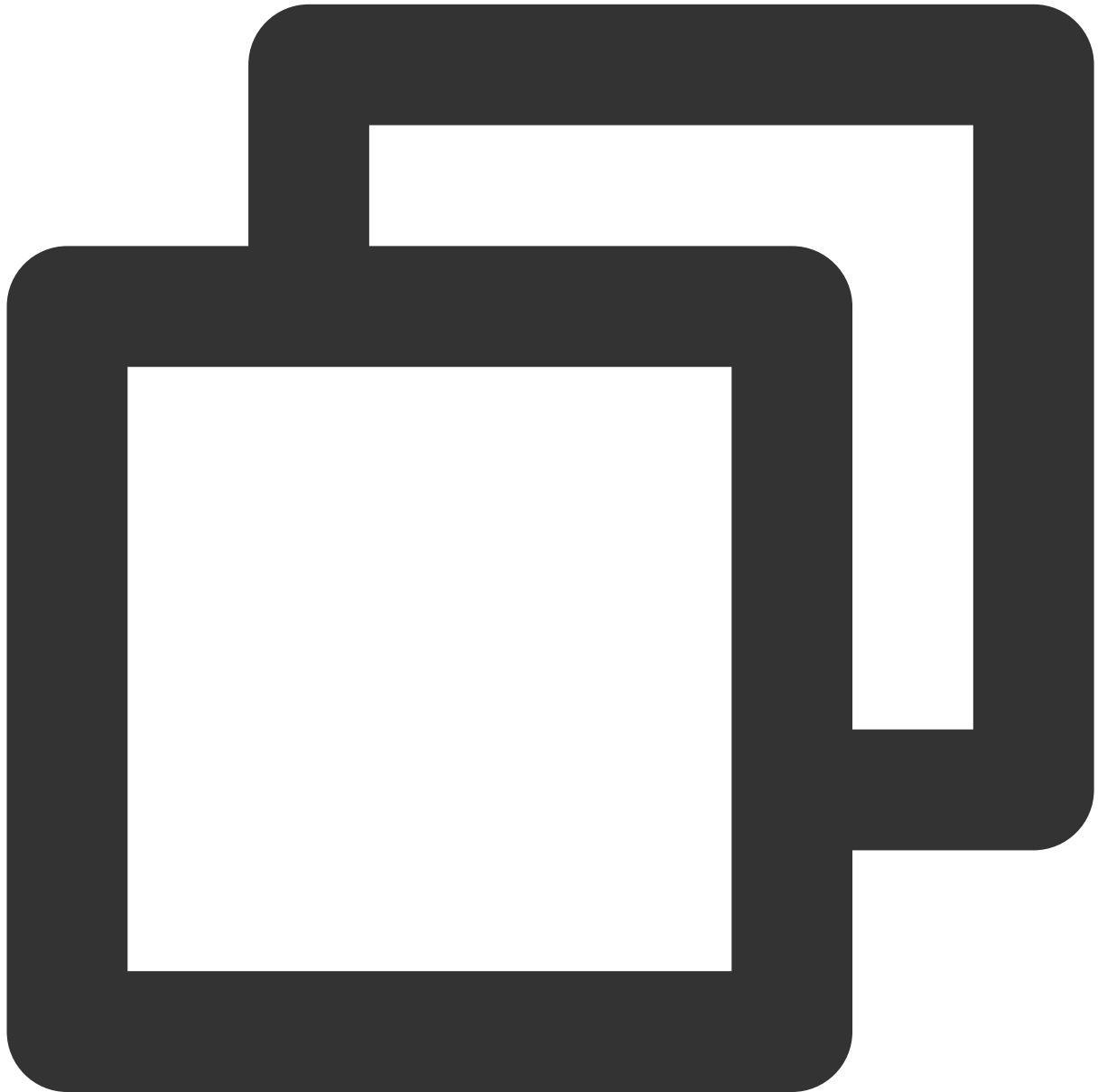
小程序打开的场景。



```
/**
 * 小程序主入口，「最近使用」列表
 */
public static final int LAUNCH_SCENE_MAIN_ENTRY = 1001;
/**
 * 扫码打开
 */
public static final int LAUNCH_SCENE_QR_CODE_FROM_SCAN = 1011;
/**
 * 搜索打开
 */
```

```
public static final int LAUNCH_SCENE_SEARCH = 2005;
```

SearchOptions

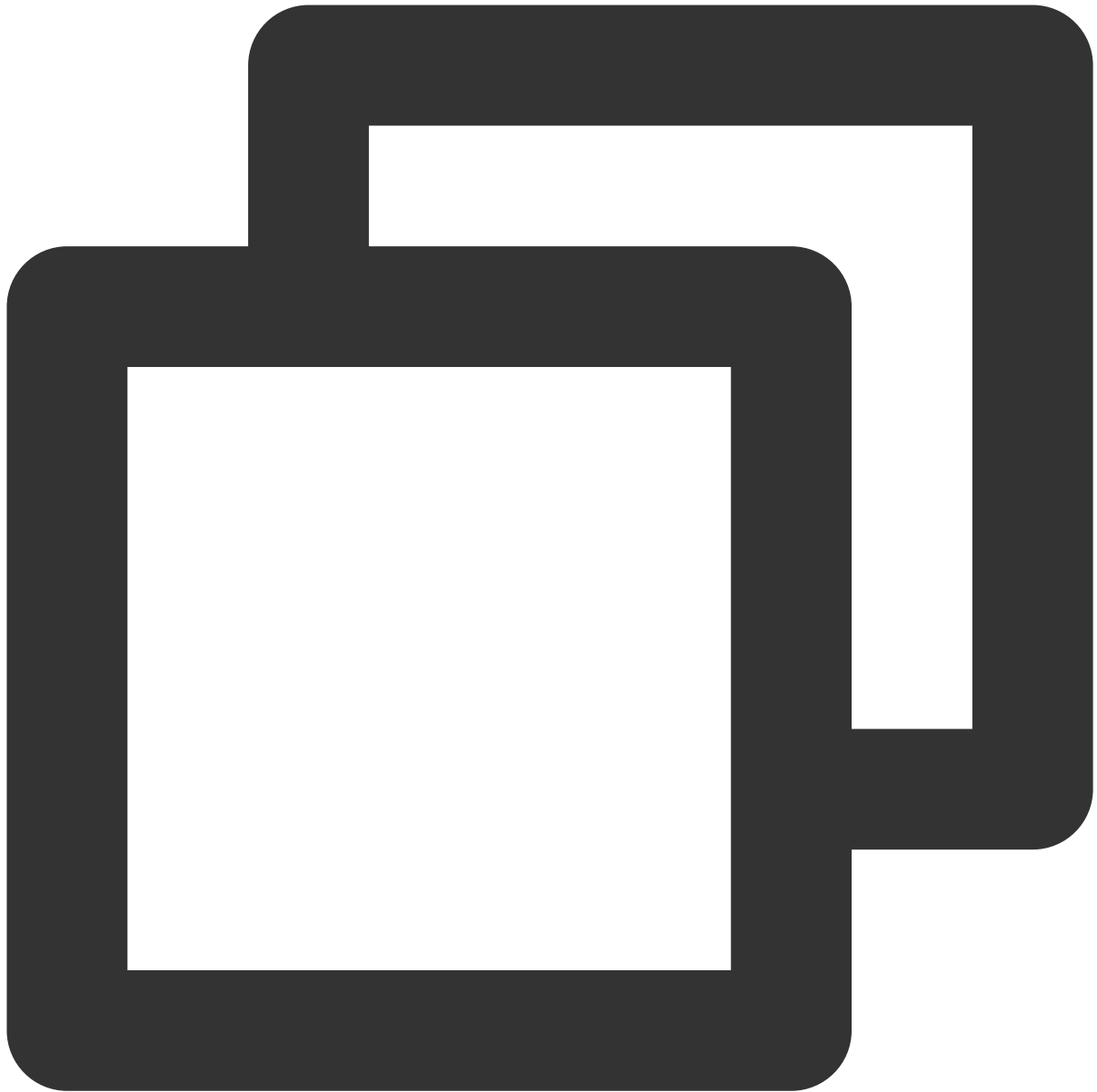


```
/**  
 * 搜索关键字，为空时搜索全部小程序  
 */  
public String keyWord = "";
```

```
/**
 * 指定一级分类
 */
public String firstLevelCate;

/**
 * 指定二级分类
 */
public String secondaryLevelCate;
```

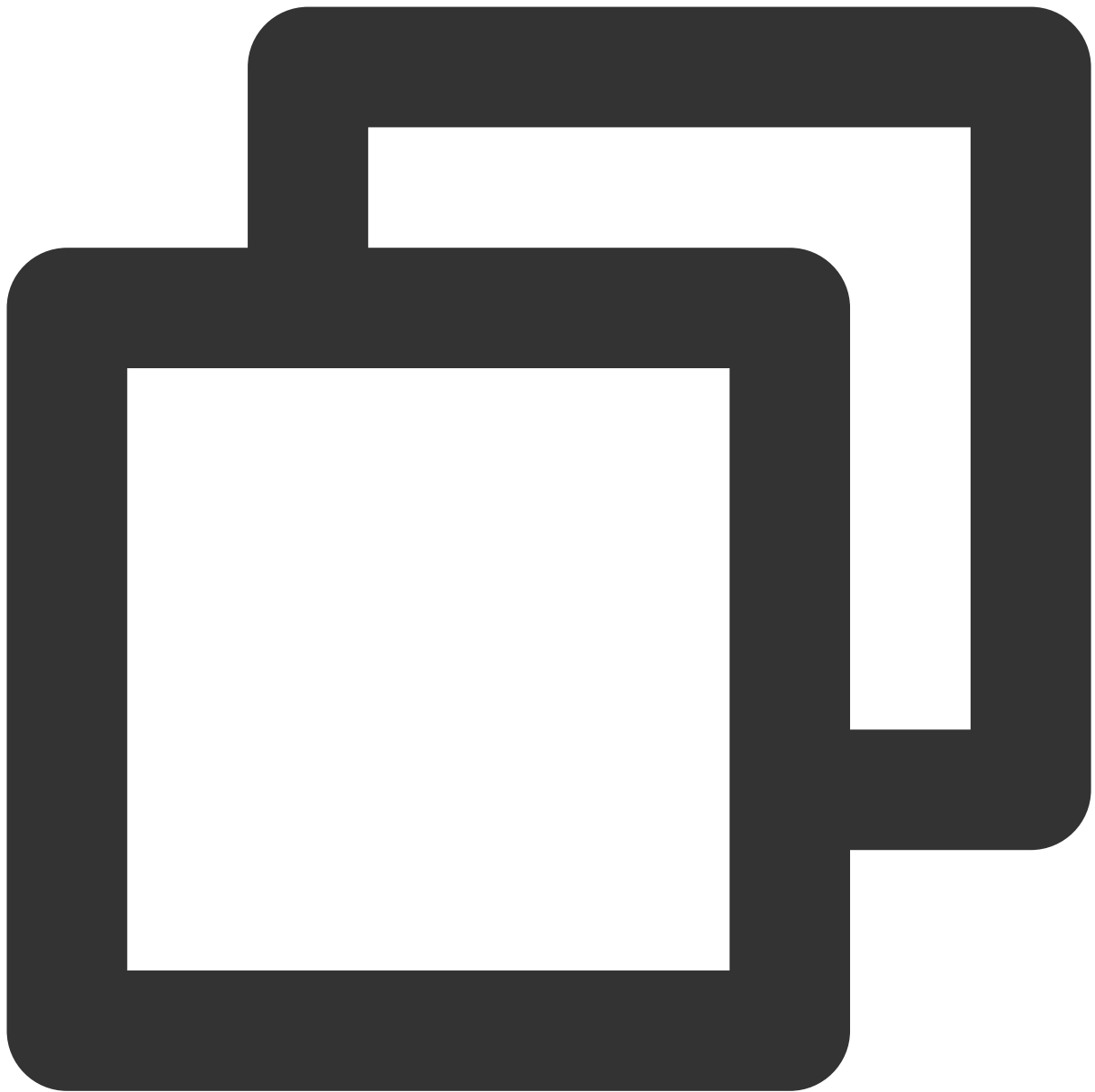
ShareData



```
/**
 * 分享来源，ShareSource中的值
 */
public int shareSource;
/**
 * 分享目标， ShareTarget中的值
 */
public int shareTarget;
/**
 * 分享面板设置的ID，用于区分分享渠道
 */
```

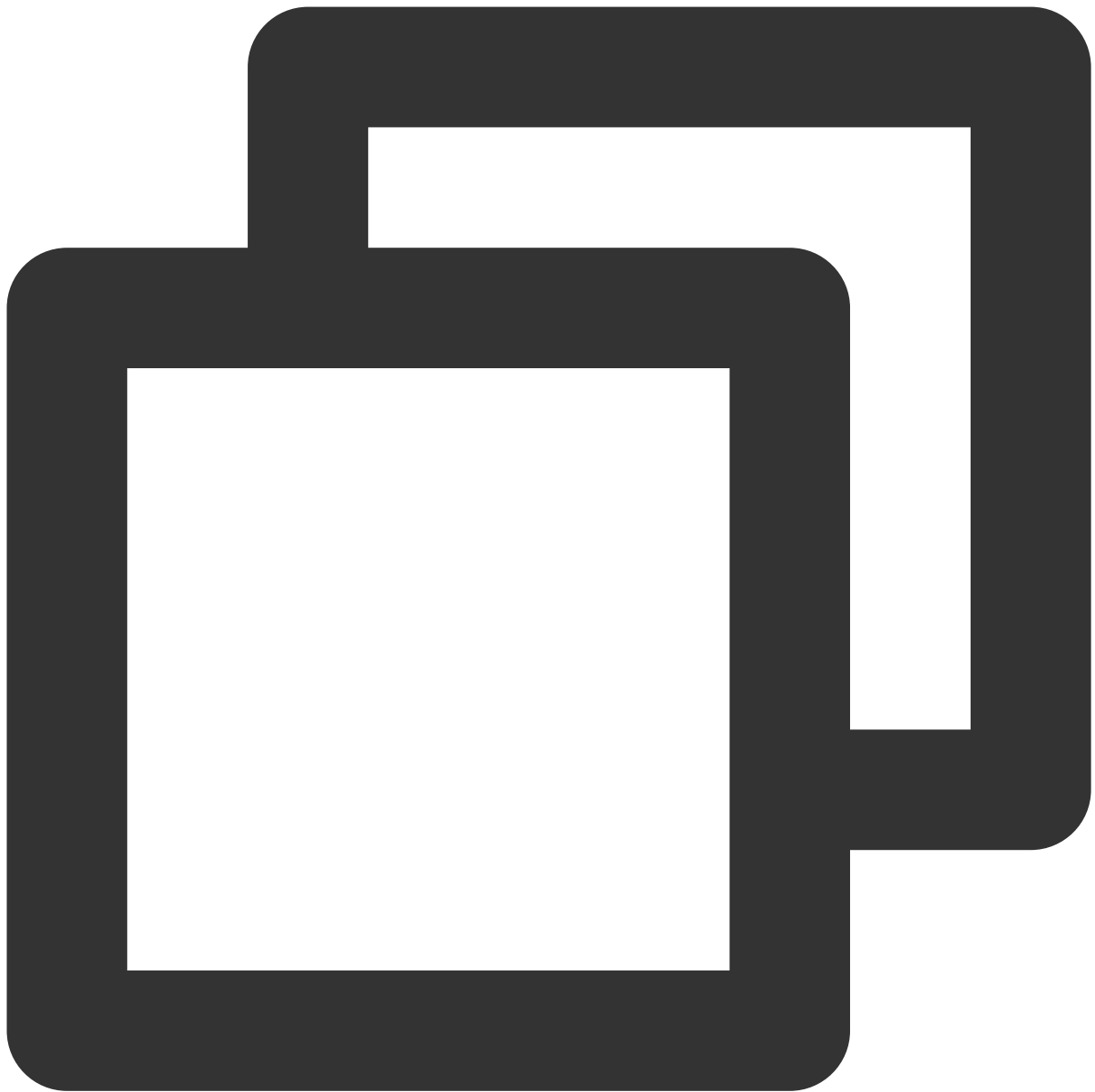
```
public int shareItemId;
/**
 * 分享标题
 */
public String title;
/**
 * 分享摘要
 */
public String summary;
/**
 * 分享图片的路径。为本地图片路径或者网络图片路径
 */
public String sharePicPath;
/**
 * 是否为本地图片。如果为True，则sharePicPath为本地图片的路径；否则，sharePicPath为网络图
 */
public boolean isLocalPic;
/**
 * 从服务端获取的字段：分享链接
 */
public String targetUrl;
/**
 * 小程序包信息
 */
protected MiniAppInfo miniAppInfo;
```

ShareSource



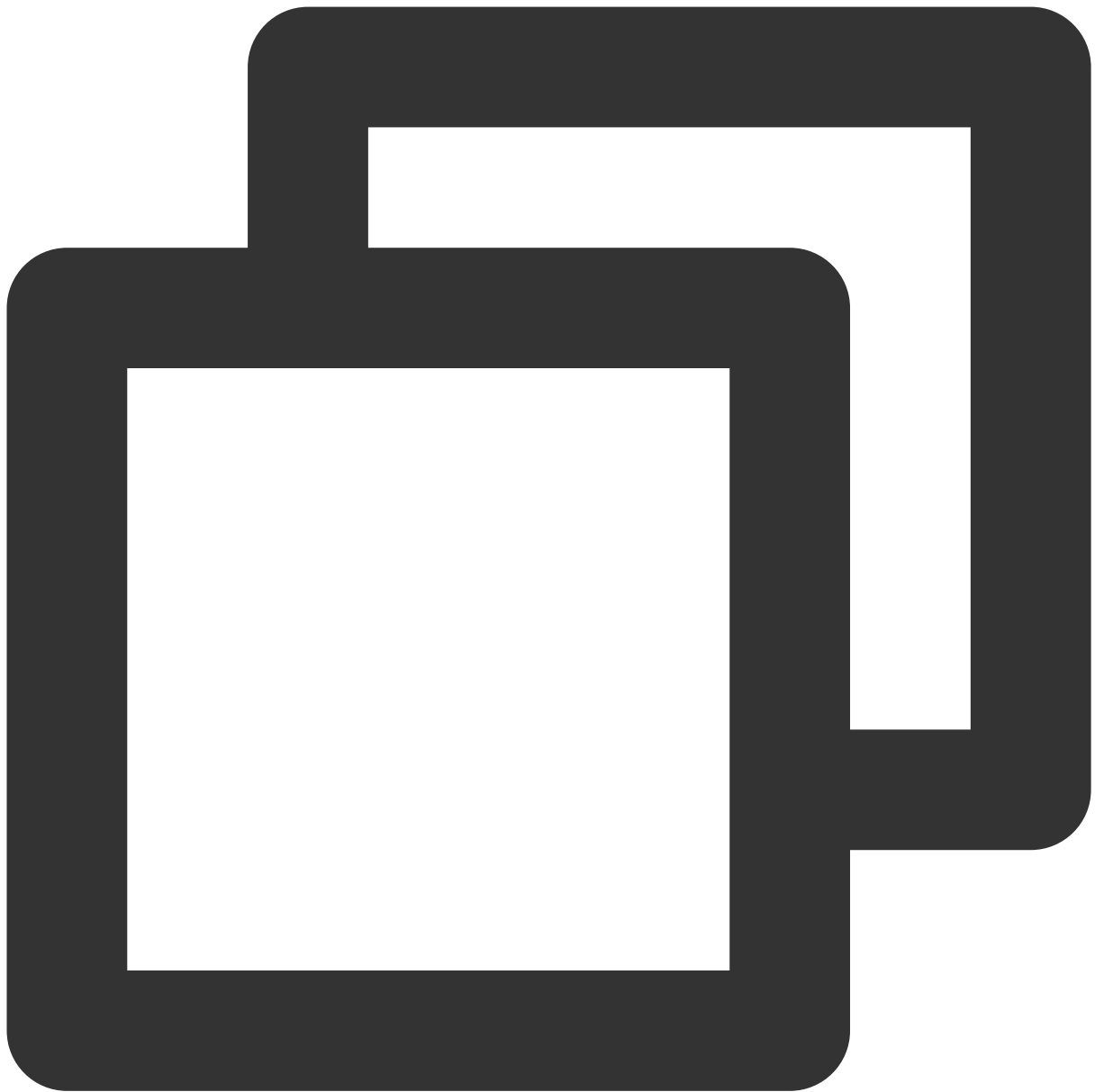
```
public static class ShareSource {  
  
    public static final int INNER_BUTTON = 11; // 来自小程序|小游戏的内部按钮  
    public static final int MORE_BUTTON = 12;  // 来自胶囊按钮的更多选项  
}
```

ShareTarget



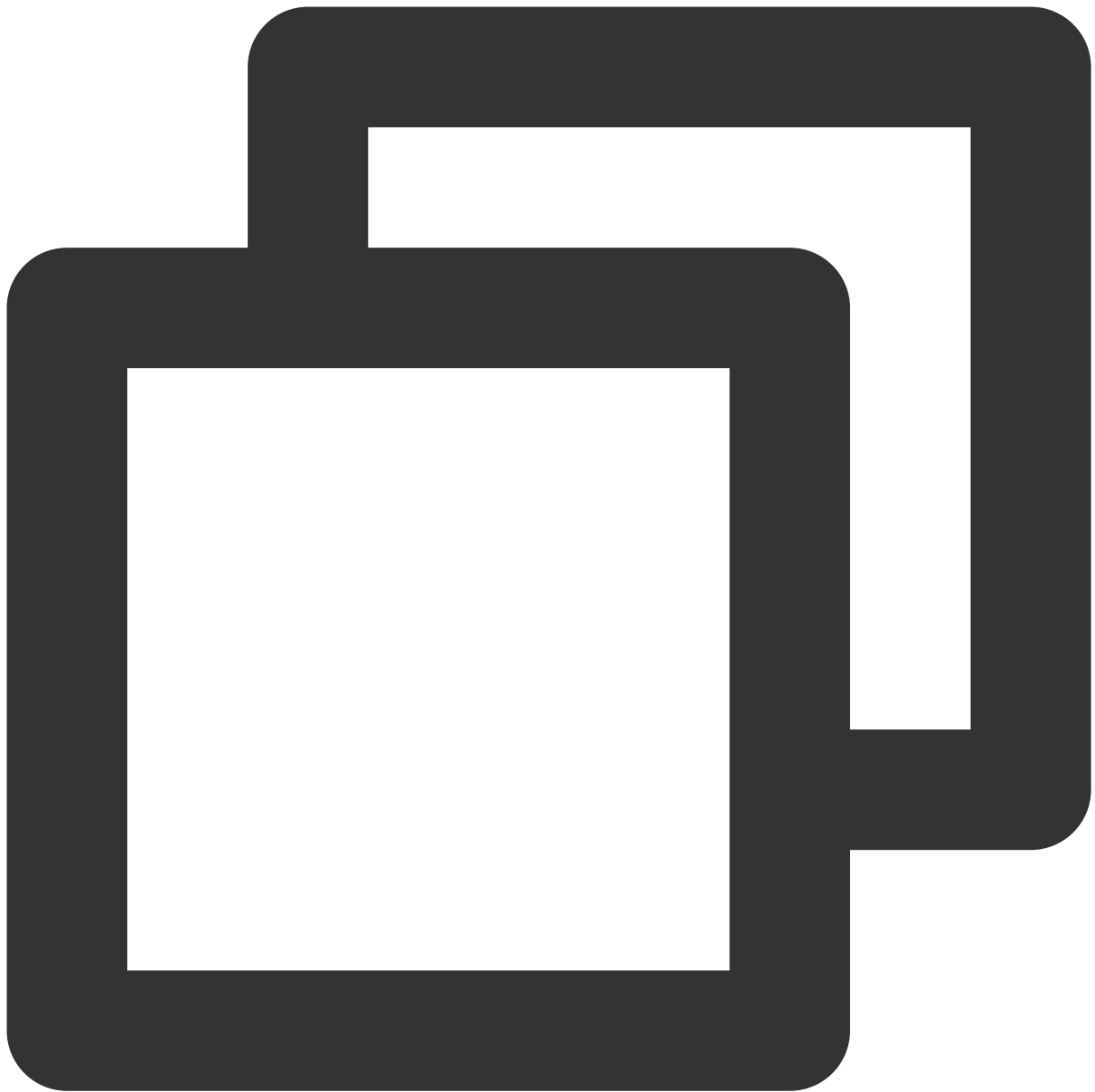
```
public static class ShareTarget {  
    public static final int QQ = 0; // 转发到QQ通讯录  
    public static final int QZONE = 1; // 转发到QQ空间  
    public static final int WECHAT_FRIEND = 3; // 转发到微信好友  
    public static final int WECHAT_MOMENTS = 4; // 转发到微信朋友圈  
}
```

ShareResult



```
public static class ShareResult {  
    public static final int SUCCESS = 0; // 分享成功  
    public static final int FAIL = 1; // 分享失败  
    public static final int CANCEL = 2; // 分享取消  
}
```

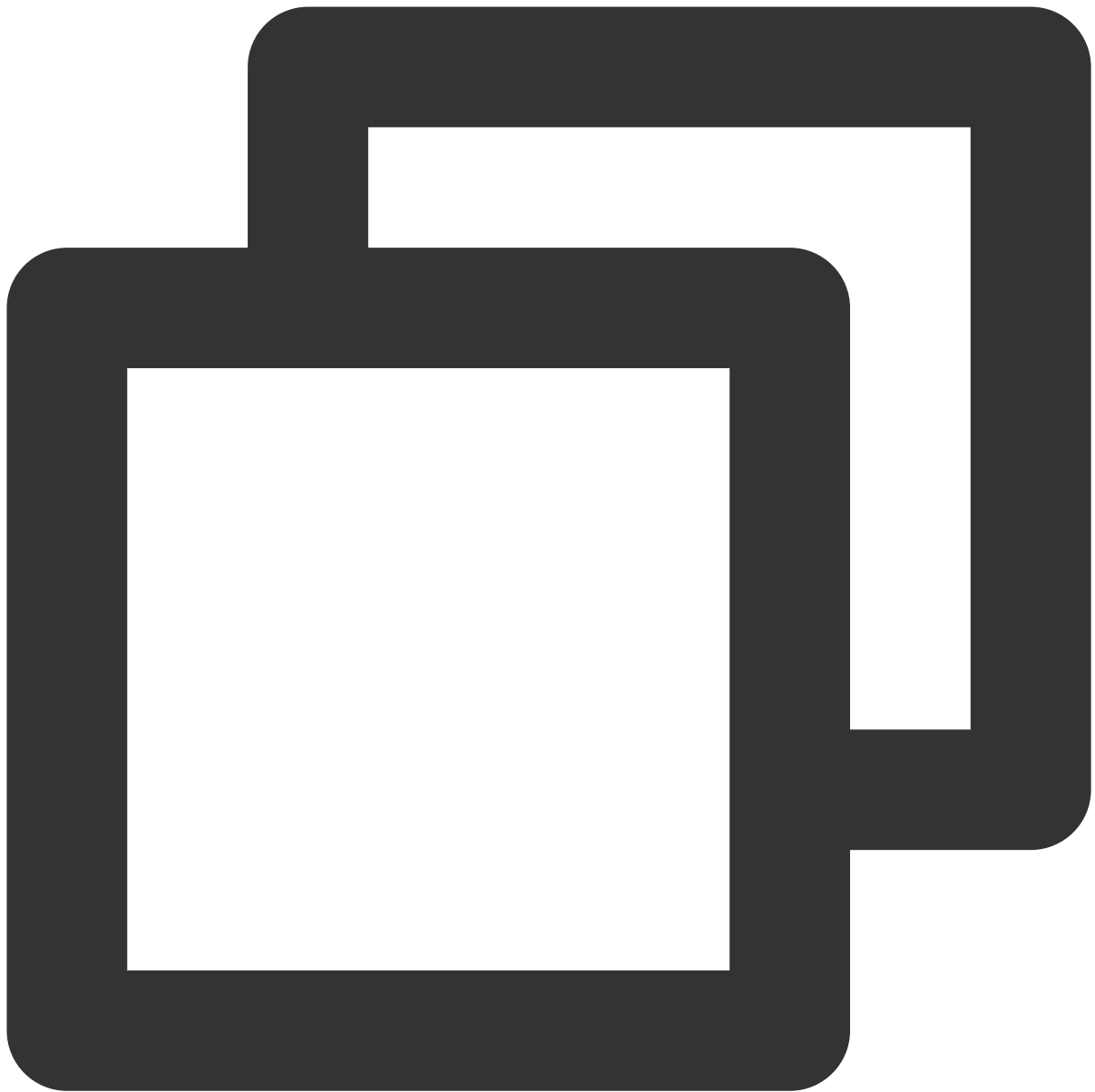
MiniStartLinkOptions



```
public class MiniStartLinkOptions {  
    /**  
     * 打开小程序时是否强制检查更新 (APP每次启动第一次打开小程序有效), false: 优先使用本地缓存, 同  
     */  
    public boolean isForceUpdate = false;  
    /**  
     * 入口地址  
     */  
    public String entryPath;  
    /**  
     * 接收小程序启动过程中错误信息  
     */  
}
```

```
    */  
    public ResultReceiver resultReceiver;  
  
    /**  
     * 小程序启动参数  
     */  
    public String params;  
}
```

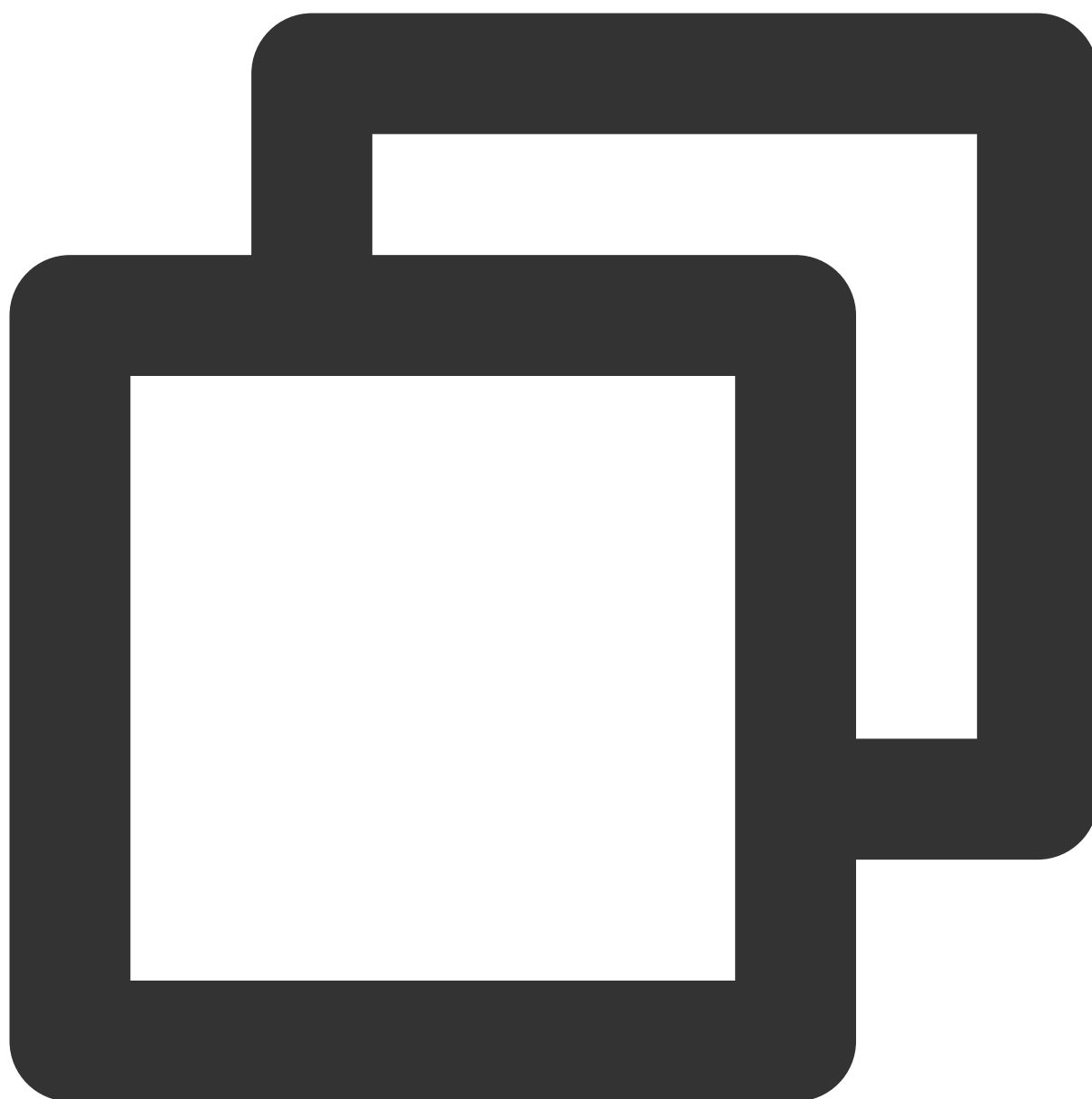
MinilnitConfig



```
/**
 * assets中配置文件名称
 */
private String configAssetName;
/**
 * 自定义配置文件路径
 */
private String configFilePath;
/**
 * 配置文件内容
```

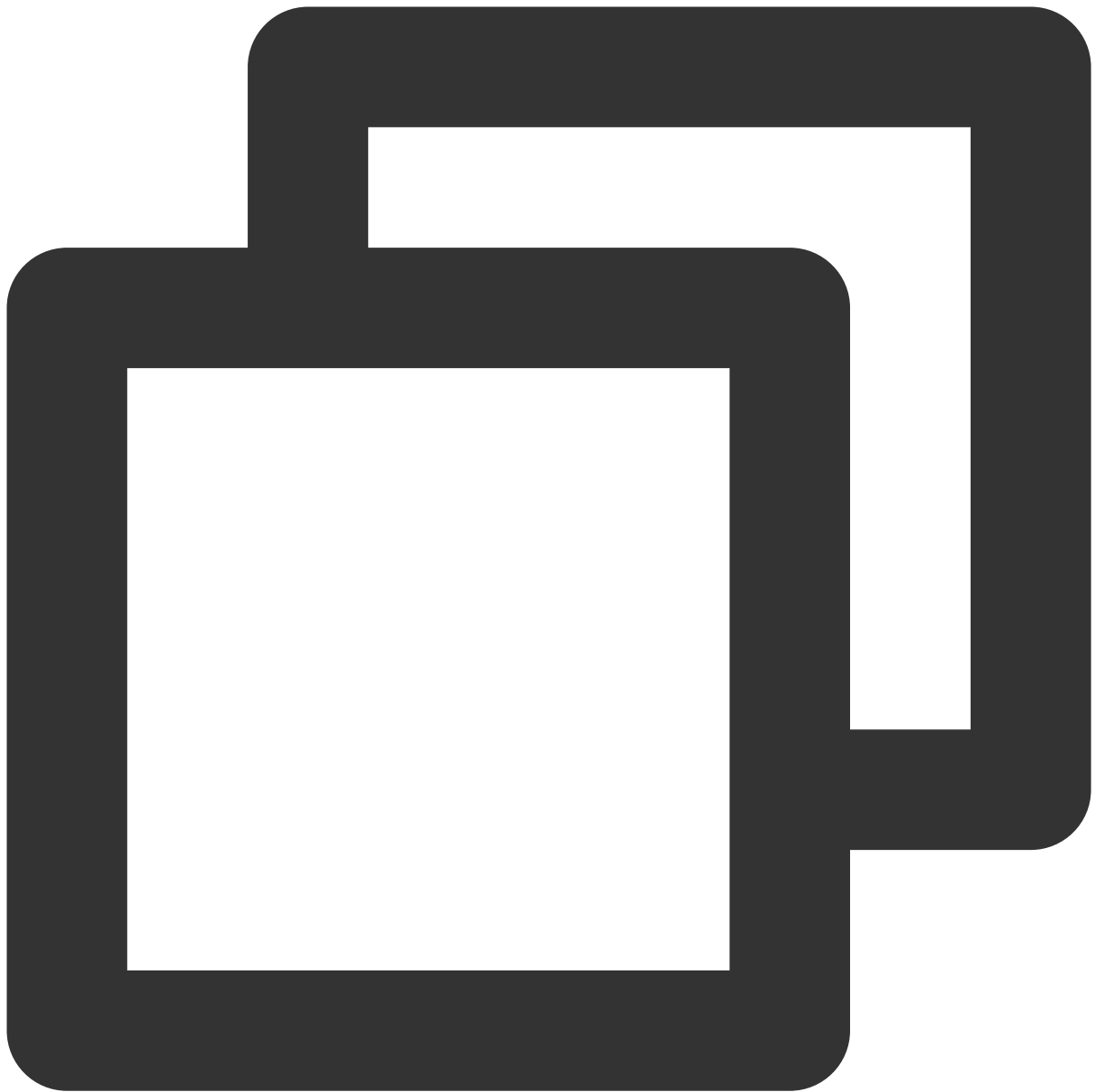
```
    */
private String configJsonStr;
/**
 * imei,用于后台小程序推送配置
 */
private String imei;
/**
 * sdk日志开关
 */
private boolean debug;
/**
 * 设置外部shark实例
 */
private IShark shark;
/**
 * 加载配置文件时，是否校验配置文件中的包名
 */
private boolean verifyPkg;
/**
 * 是否使用x5内核
 */
private boolean isUserX5Core = true;
/**
 * 是否强制使用内核基础库
 */
private boolean forceUseBaseLibInAsset;
/**
 * 预置离线包assets路径
 */
private String assetPathOfPresets;
```

IMiniAppContext



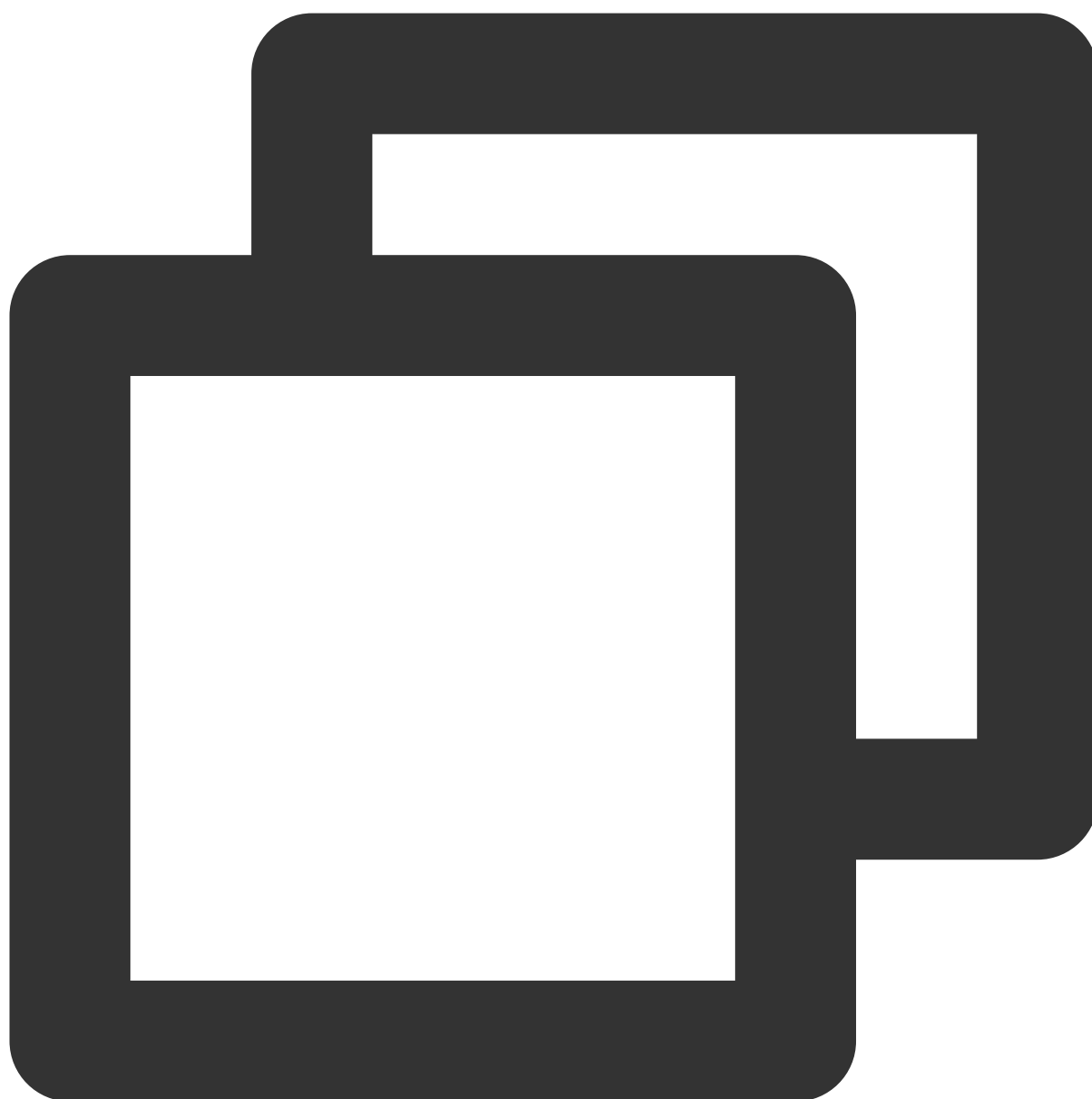
```
/**  
 * 返回小程序信息  
 */  
MiniAppInfo getMiniAppInfo();
```

MiniAppInfo



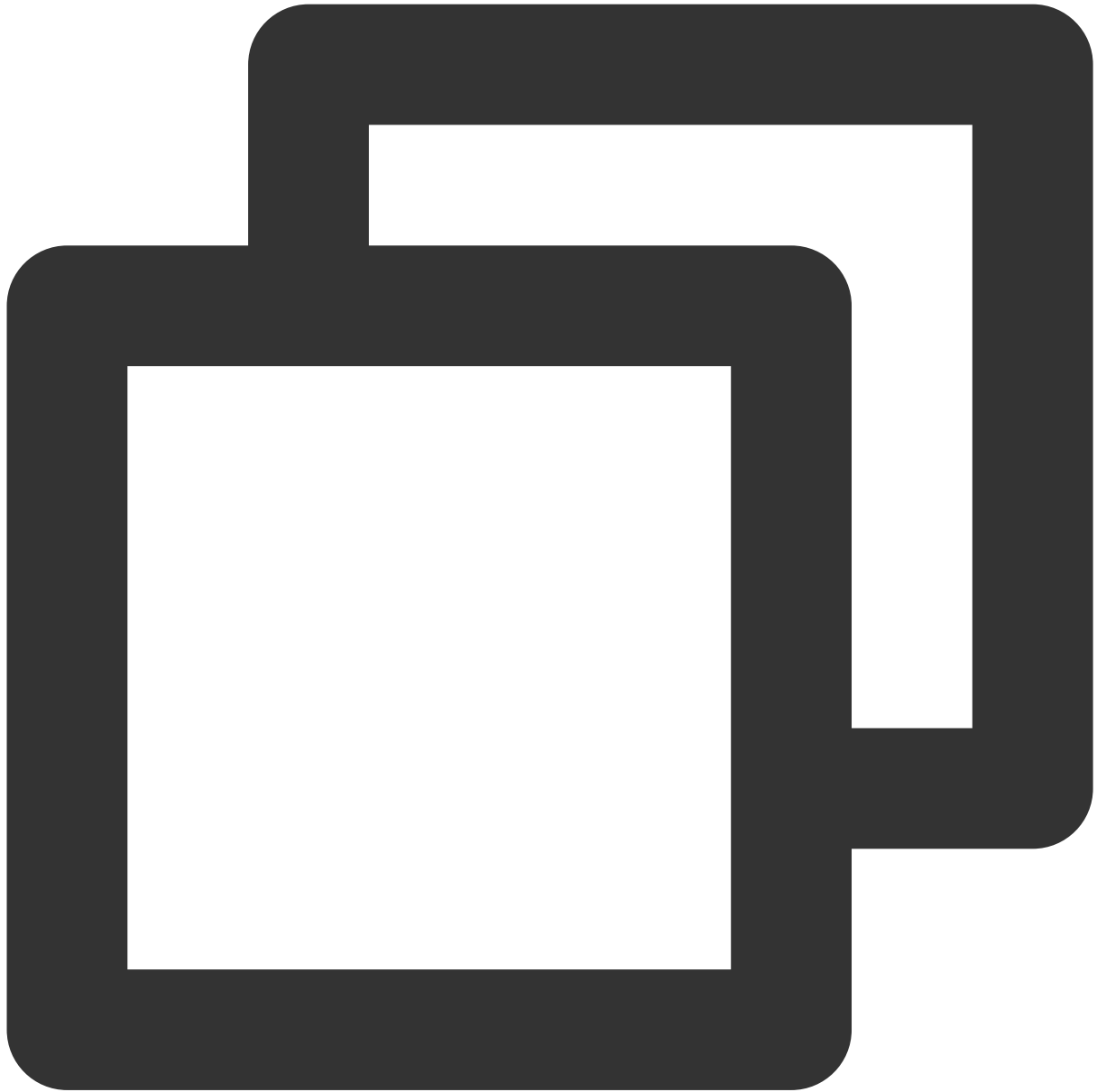
```
public String appId;//小程序对应APPID  
public String name;//小程序的名称  
public String iconUrl;//小程序图标url  
public String version;//小程序版本号  
public int verType;//小程序类型：开发、预览、正式版
```

IpcCallback



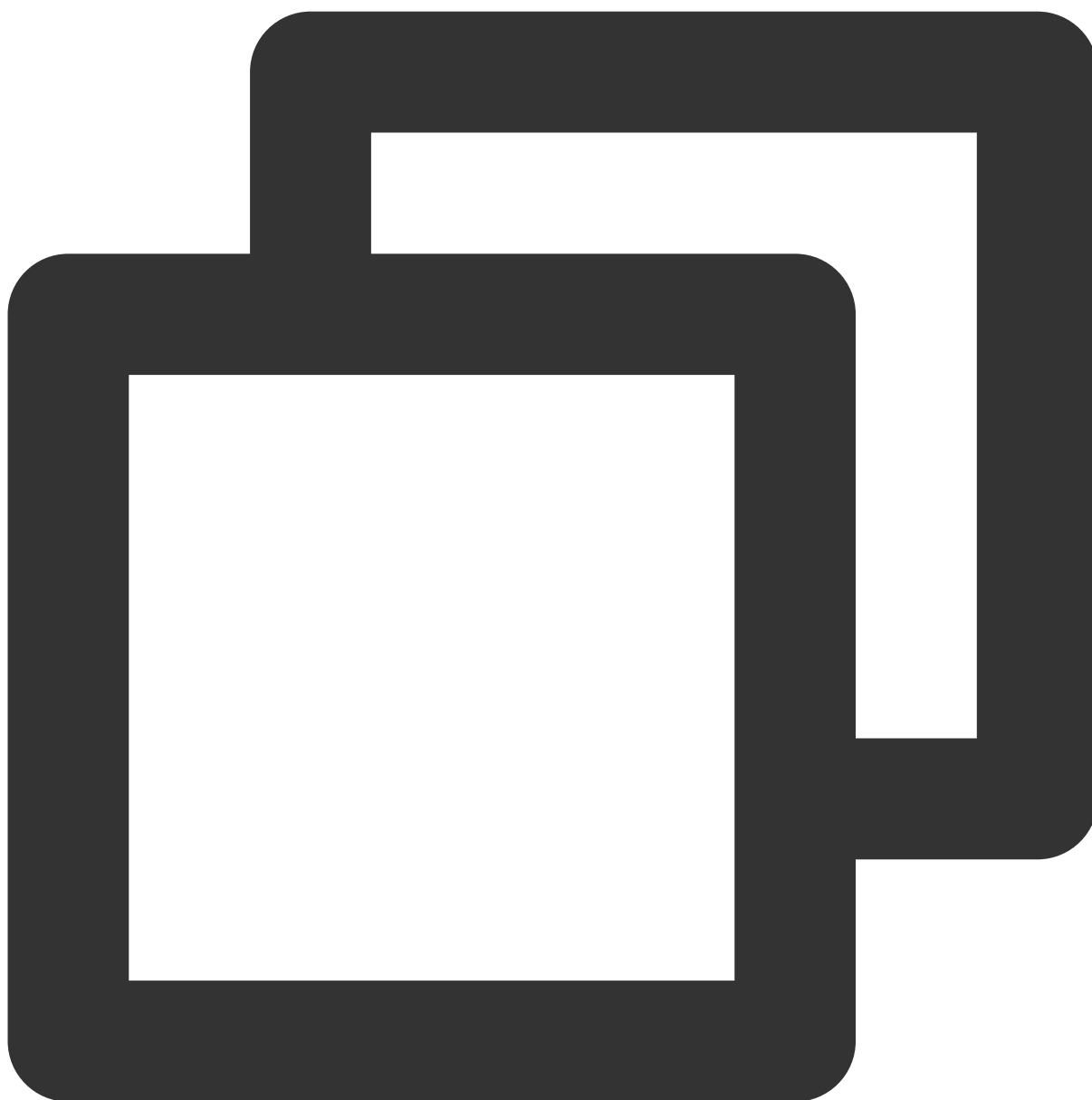
```
public interface IpcCallback {  
    /**  
     * 进程通信回调  
     * @param isSuccess 是否成功调用  
     * @param response 返回数据  
     */  
    void result(boolean isSuccess, Bundle response);  
}
```

IpcRequestEvent



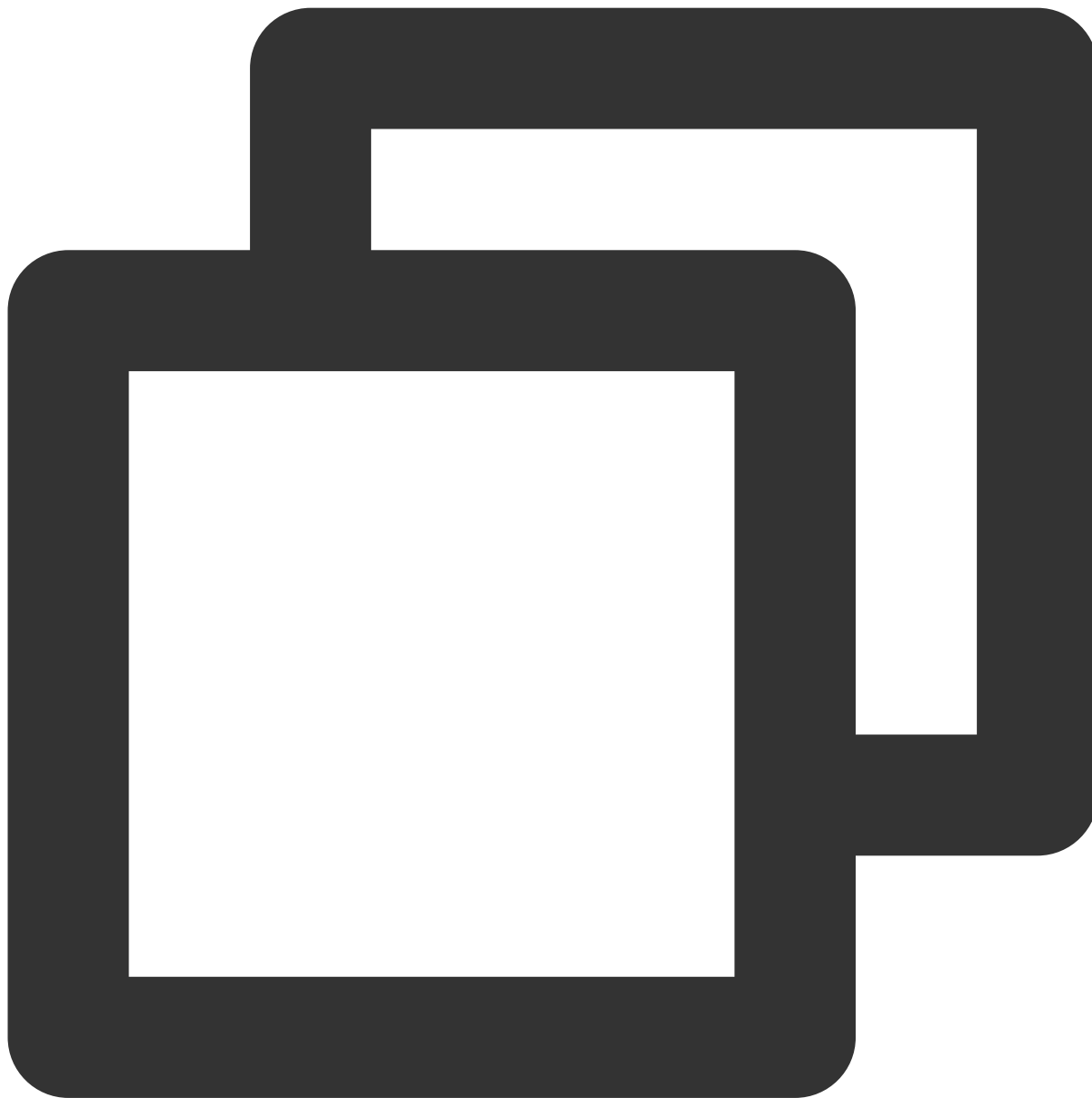
```
public Context context;  
//数据  
public Bundle data;  
//返回回调  
public IpcCallback callback;
```

RequestEvent



```
//小程序activity
public WeakReference<Activity> activityRef;
//事件名
public String event;
//事件参数
public String jsonParams;
```

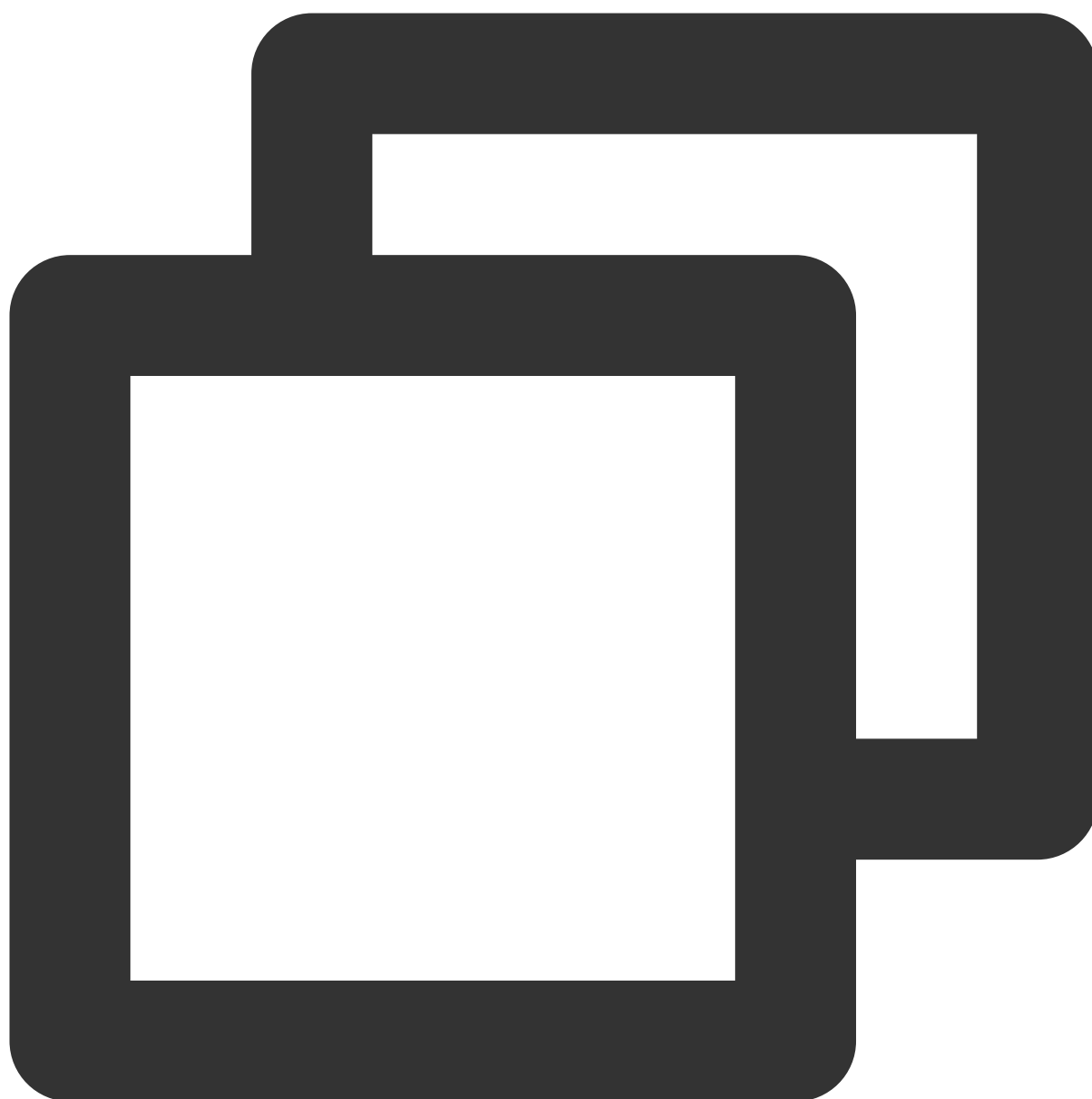
AppState



```
/**
 * 小程序启动
 */
int STATE_START = 1;
/**
 * 小程序切换到前台
 */
int STATE_FOREGROUND = 2;
```

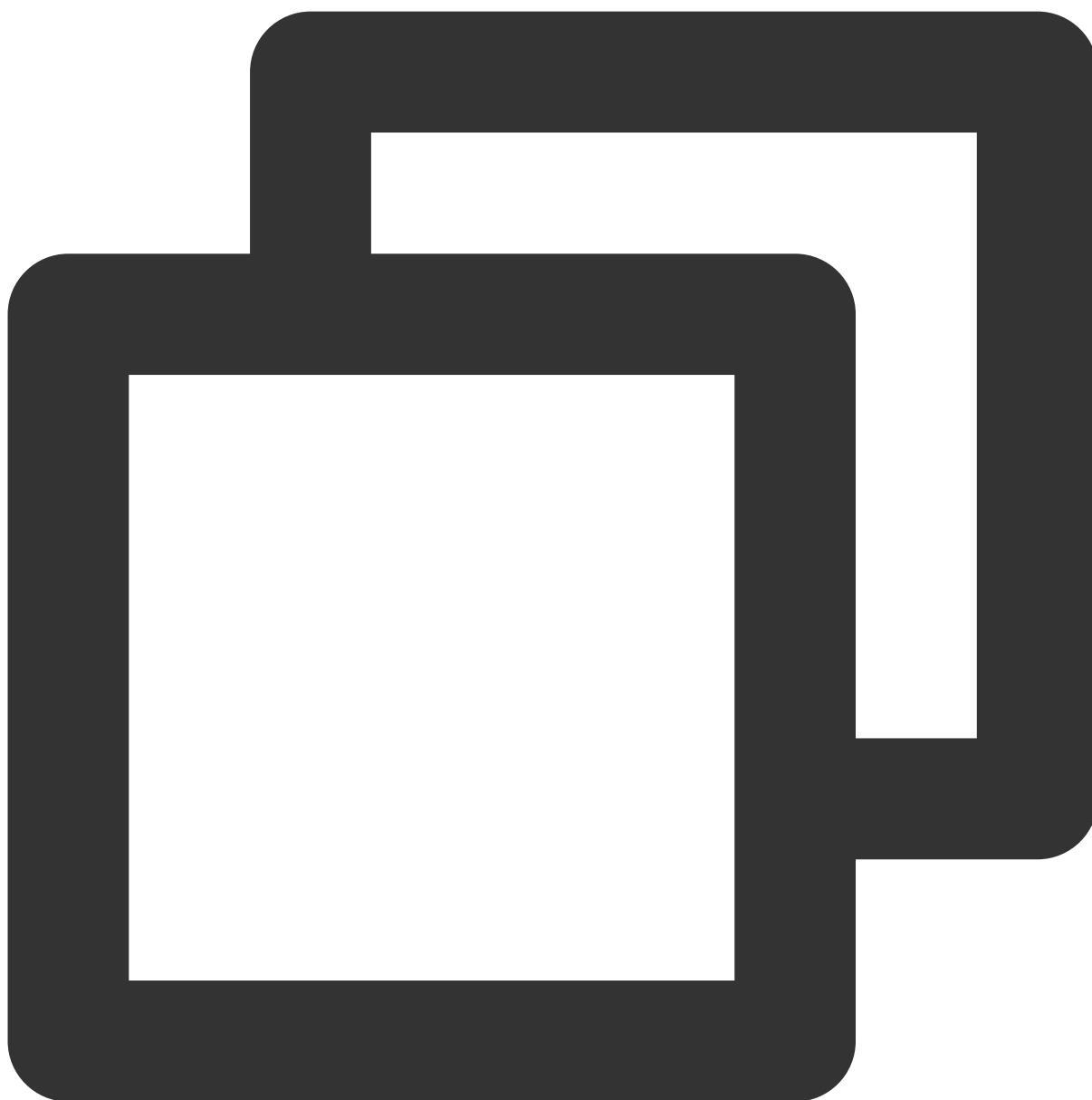
```
/**
 * 胶囊关闭小程序
 */
int STATE_CLOSE = 3;
/**
 * 小程序切换到后台
 */
int STATE_BACKGROUND = 4;
/**
 * 小程序销毁
 */
int STATE_DESTROY = 5;
```

MiniAppEvent



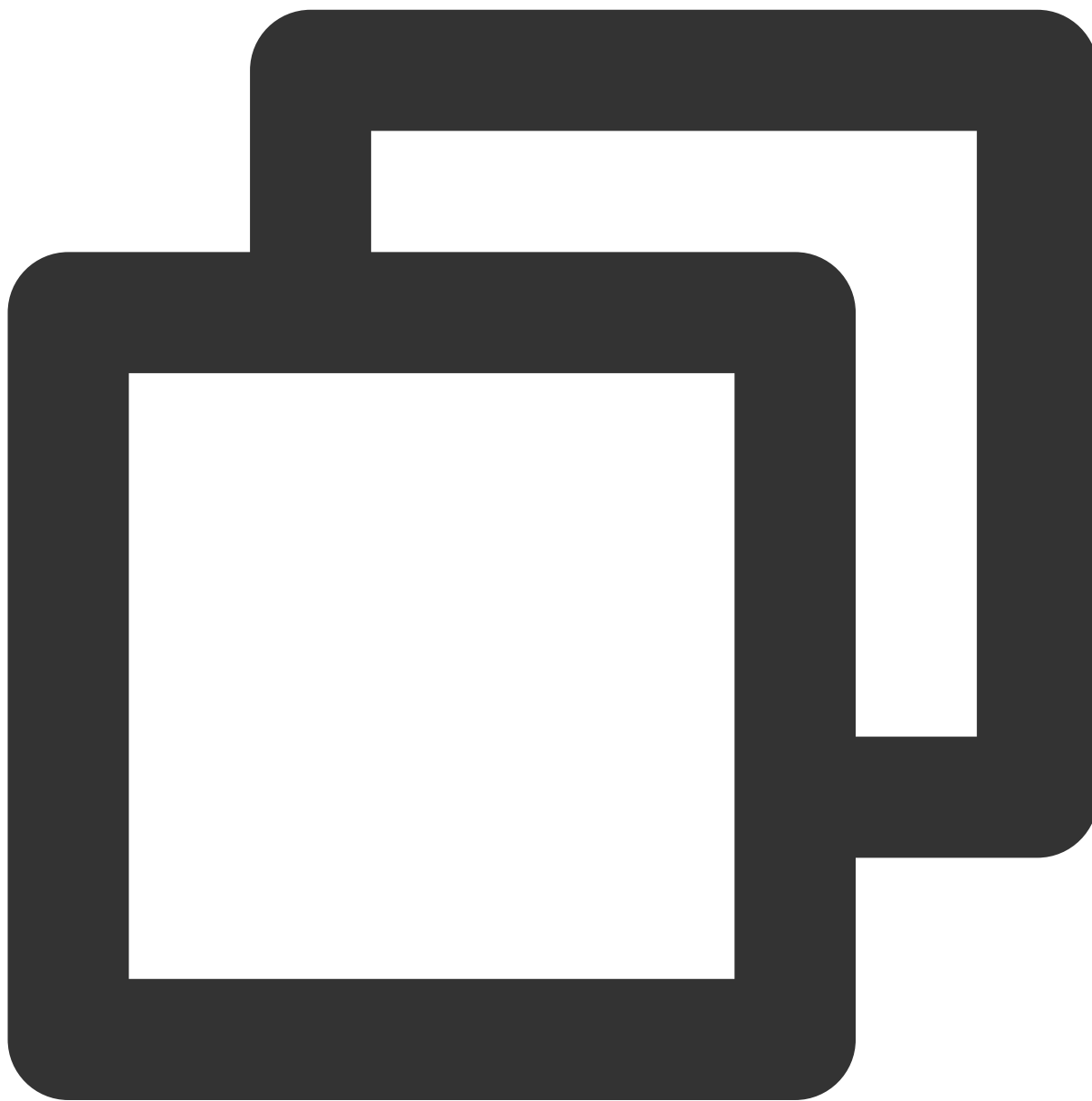
```
/**
 * 小程序信息
 */
public MiniApp miniApp;
/**
 * 是否热启动
 */
public boolean isHotStart;
```

PreDownloadInfo



```
/**
 * 小程序appId
 */
public String appId;
/**
 * 是否下载小程序包
 */
public boolean isDownload;
```

IDownloadCallback



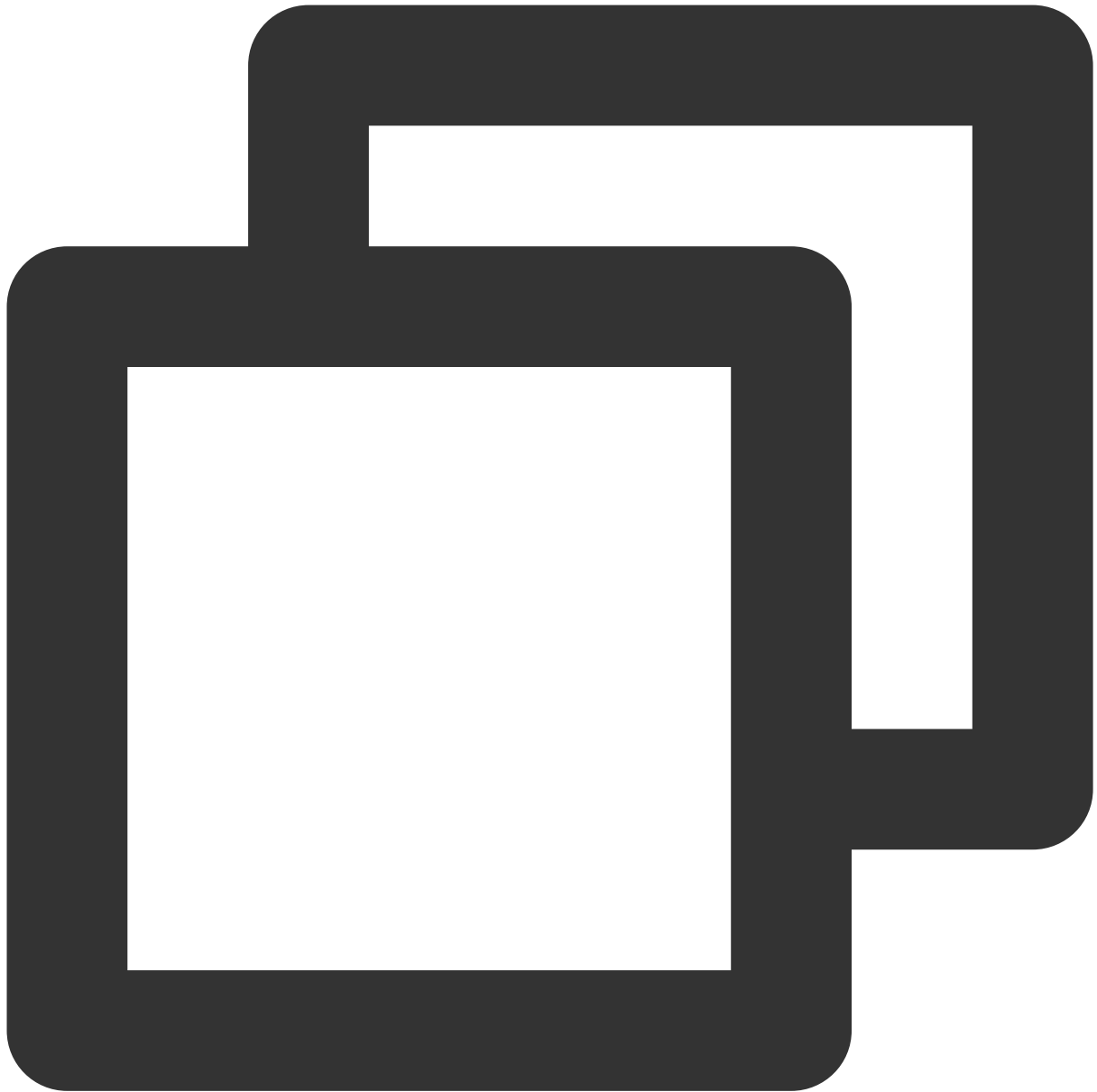
```
/**  
 * 下载成功回调  
 *  
 * @param downloadInfo  
 */
```



```
void onFinish(DownloadInfo downloadInfo);

/**
 * 回调失败
 *
 * @param downloadInfo
 */
void onError(DownloadInfo downloadInfo);
```

DownloadInfo

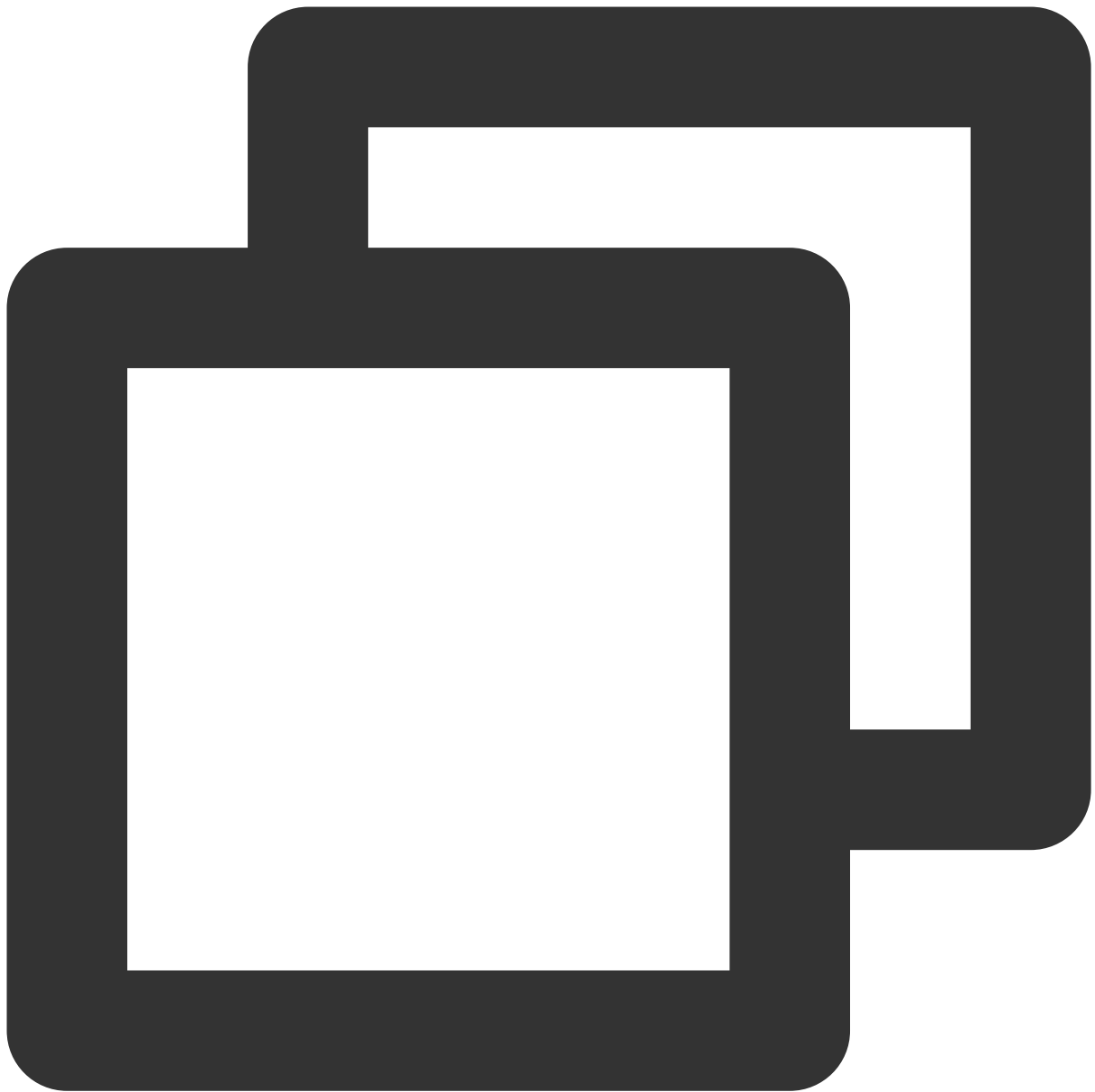


```
//下载IOException和IllegalAccessException异常
public static final int CODE_DOWNLOAD_IOEXCEPTION = -100001;
//下载Exception异常
public static final int CODE_DOWNLOAD_EXCEPTION = -100002;
//没有网
public static final int CODE_NO_NETWORK = -100003;
//下载参数错误
public static final int CODE_PARAM_ERROR = -100004;
//下载目录创建失败
public static final int CODE_DOWNLOAD_DIR_CREATE_FAIL = -100005;
```

```
//下载目录创建失败
public static final int CODE_MINI_APP_PARSE_FAIL = -12013;

/**
 * 小程序 id
 */
private String appId;
/**
 * 错误 code
 */
private int errCode;
/**
 * 错误msg
 */
private String message;
```

IMiniAppFileManager

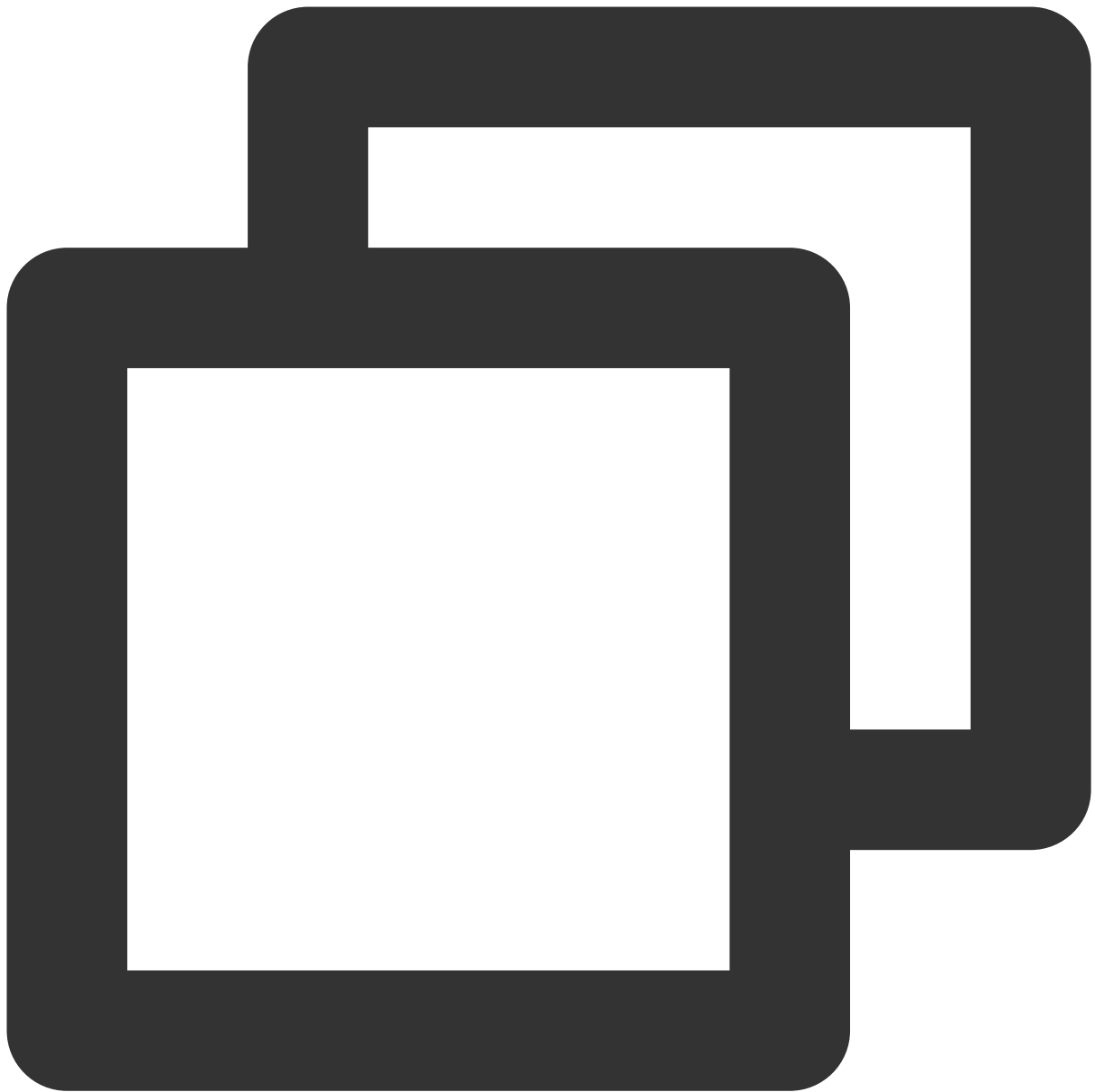


```
/**
 * 获取wxfile绝对路径
 * @param wxFilePath
 * @return
 */
String getAbsolutePath(String wxFilePath);

/**
 * 将绝对路径转为 wxfile
 * @param path
 * @return
 */
```

```
*/  
String getWxFilePath(String path);  
  
/**  
 * 获取临时目录  
 * @param suffix  
 * @return  
 */  
String getTmpPath(String suffix);
```

BaseJsPlugin



```
/**
 * 小程序上下文
 */
protected IMiniAppContext mMiniAppContext;
/**
 * 小程序信息
 */
protected MiniAppInfo mMiniAppInfo;
/**
 * 小程序包信息
 */
```

```
protected ApkInfo mApkgInfo;
```

扩展组件 SDK

最近更新时间：2024-06-27 16:46:56

小程序的主 SDK(mini_core) 提供了小程序运行时基本能力，除 mini_core 以外，小程序 SDK 还提供如下的扩展组件，宿主应用可以根据需要进行接入。

同层渲染扩展 SDK

说明：

小程序 SDK 的同层渲染能力依赖于腾讯 X5 内核，为了实现同层渲染的能力，需要宿主应用集成同层渲染的扩展组件。

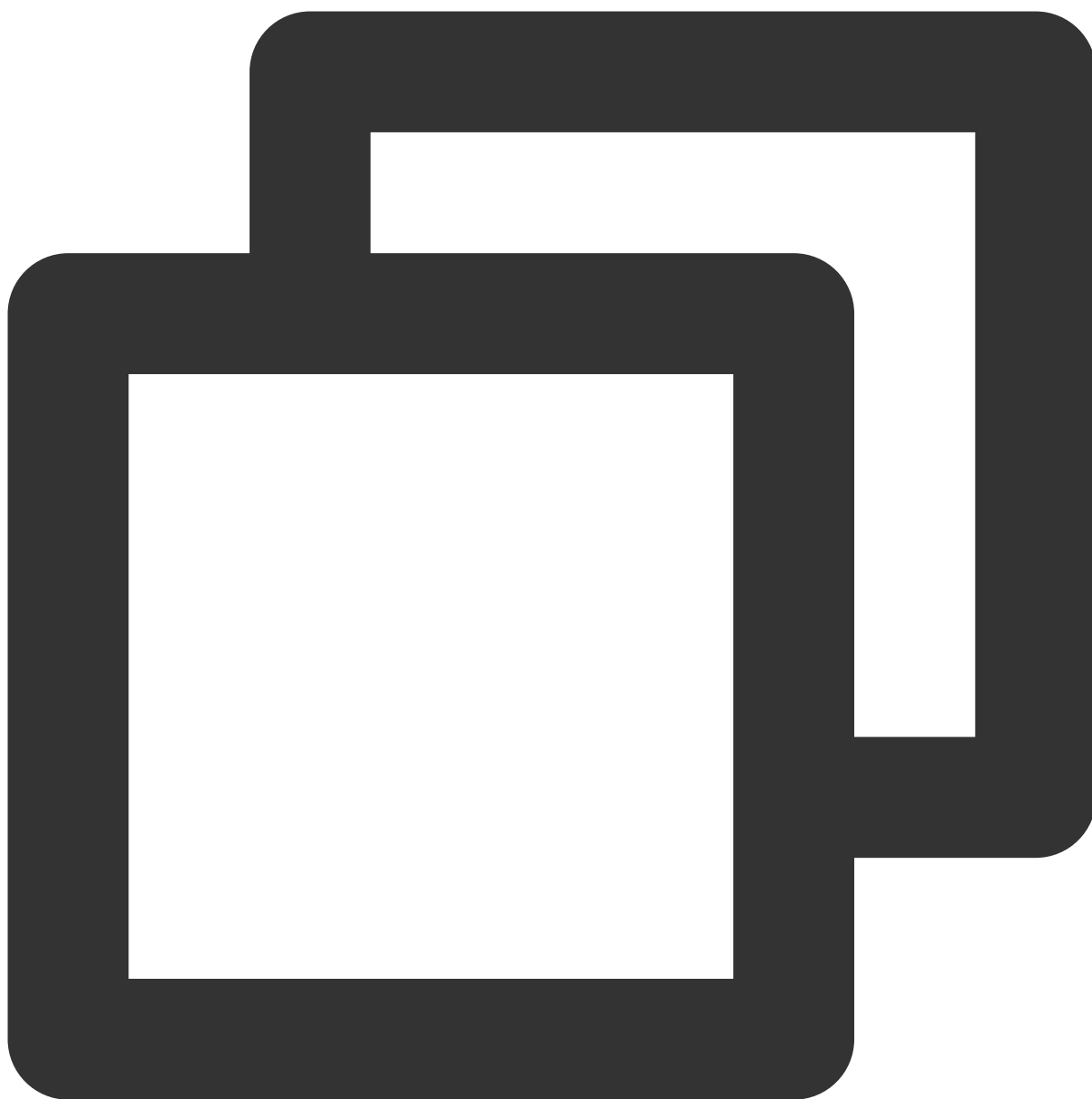
如果开发者原来的工程中已集成了腾讯X5内核，小程序 SDK 支持多种版本X5内核，客户需根据自身工程实际情况进行选择。

注意：

集成小程序 SDK 后开发者切记不要自己调用X5的初始化，因为小程序 SDK 会进行初始化，如果开发者自己调用初始化可能导致X5初始化失败，进而影响小程序同层渲染能力。

公网版本 X5 内核

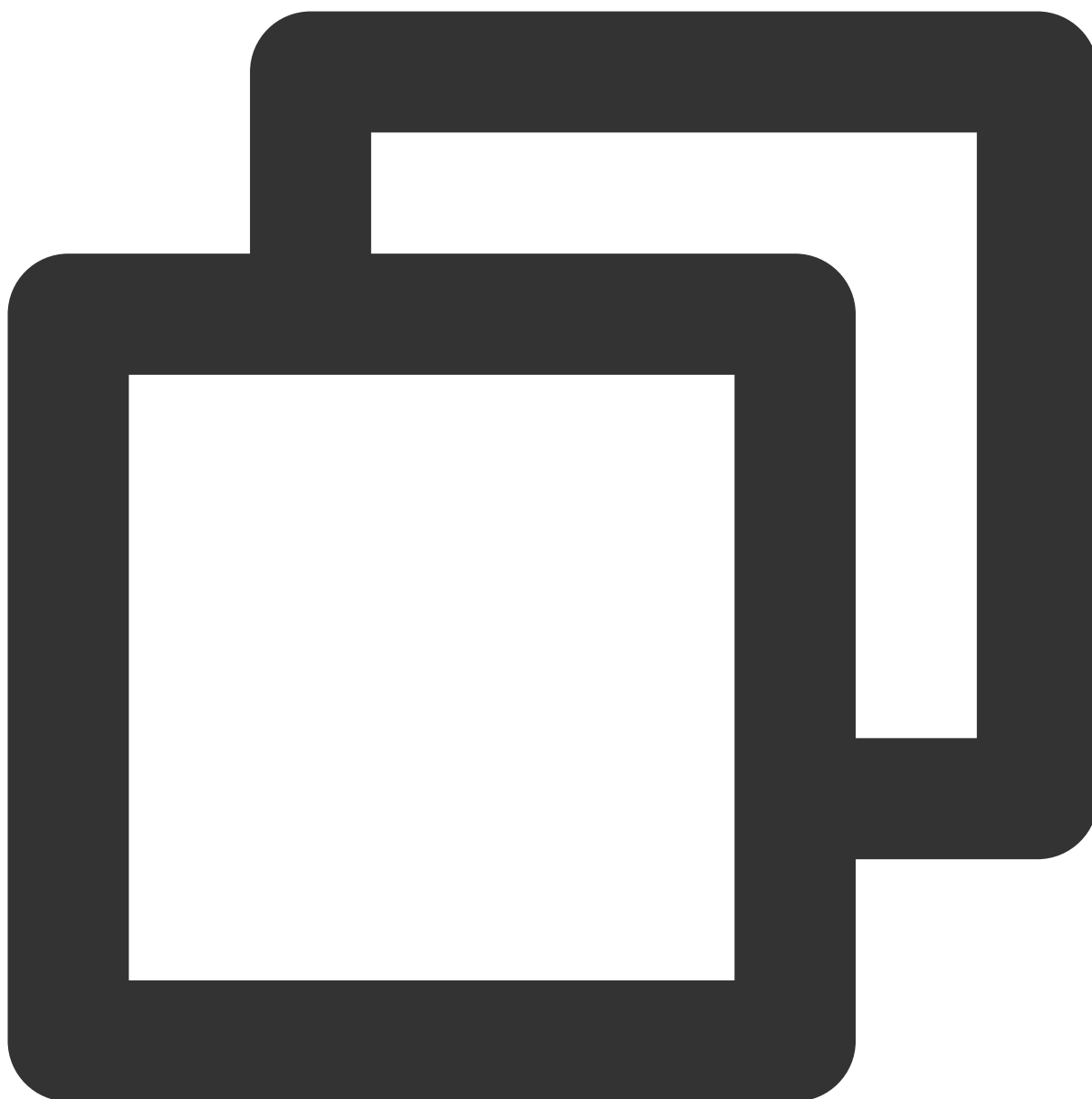
公网版本 X5 内核为腾讯 X5对外提供使用的内核，它属于共享内核（共享微信、QQ 等内核），且会访问腾讯相关后台服务。



```
implementation 'com.tencent.tbs:tbssdk:${version}' //版本信息请参考 Android 更新日志  
implementation 'com.tencent.tcmpp.android:mini_extra_public_x5:${version}' //版本信息
```

静态内核

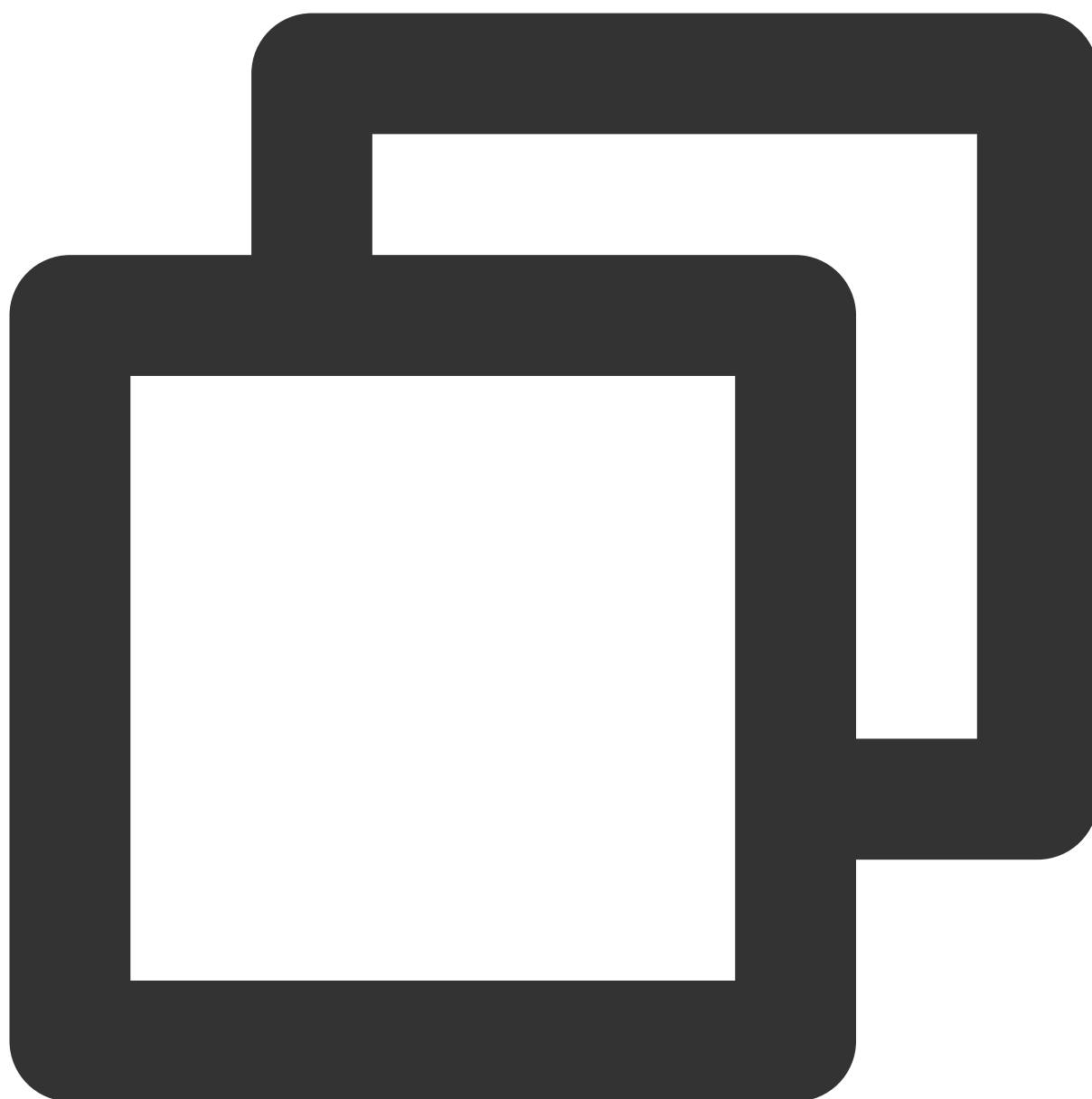
静态 x5 内核将整个内核打包的 aar 中，同时支持 armeabi、arm64-v7a、arm64-v8a 多种架构，但 aar 包体积比较大；并支持小程序同层渲染。



```
//在工程中引入依赖
implementation 'com.tencent.tmf.android:tbscore:${version}' //版本信息请参考 Android
implementation 'com.tencent.tcmpp.android:mini_extra_static_x5_new:${version}' //版

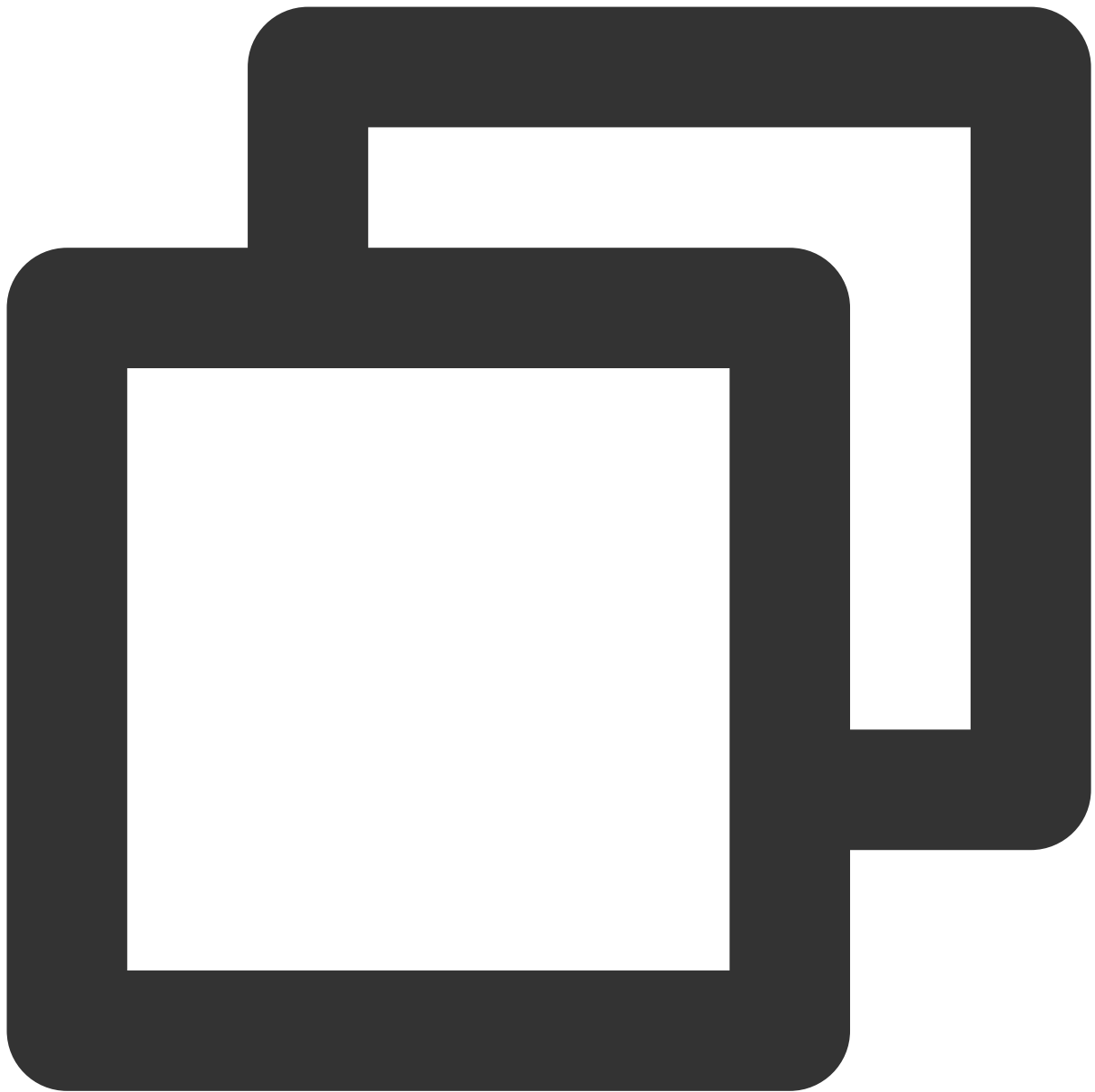
//工程的application中开启extractNativeLibs配置，必须设置为true，否则内核初始化失败
<application
    android:extractNativeLibs="true">
</application>
```

开发者选择过滤打包后的 apk 支持的架构，来减小 apk 的大小。



```
ndk { abiFilters "armeabi" "armeabi-v7a" "arm64-v8a" }
```

通过 `MiniInitConfig` 配置内核 `LicenseKey`，配置代码如下（`LicenseKey` 需要找相关人员进行申请）。

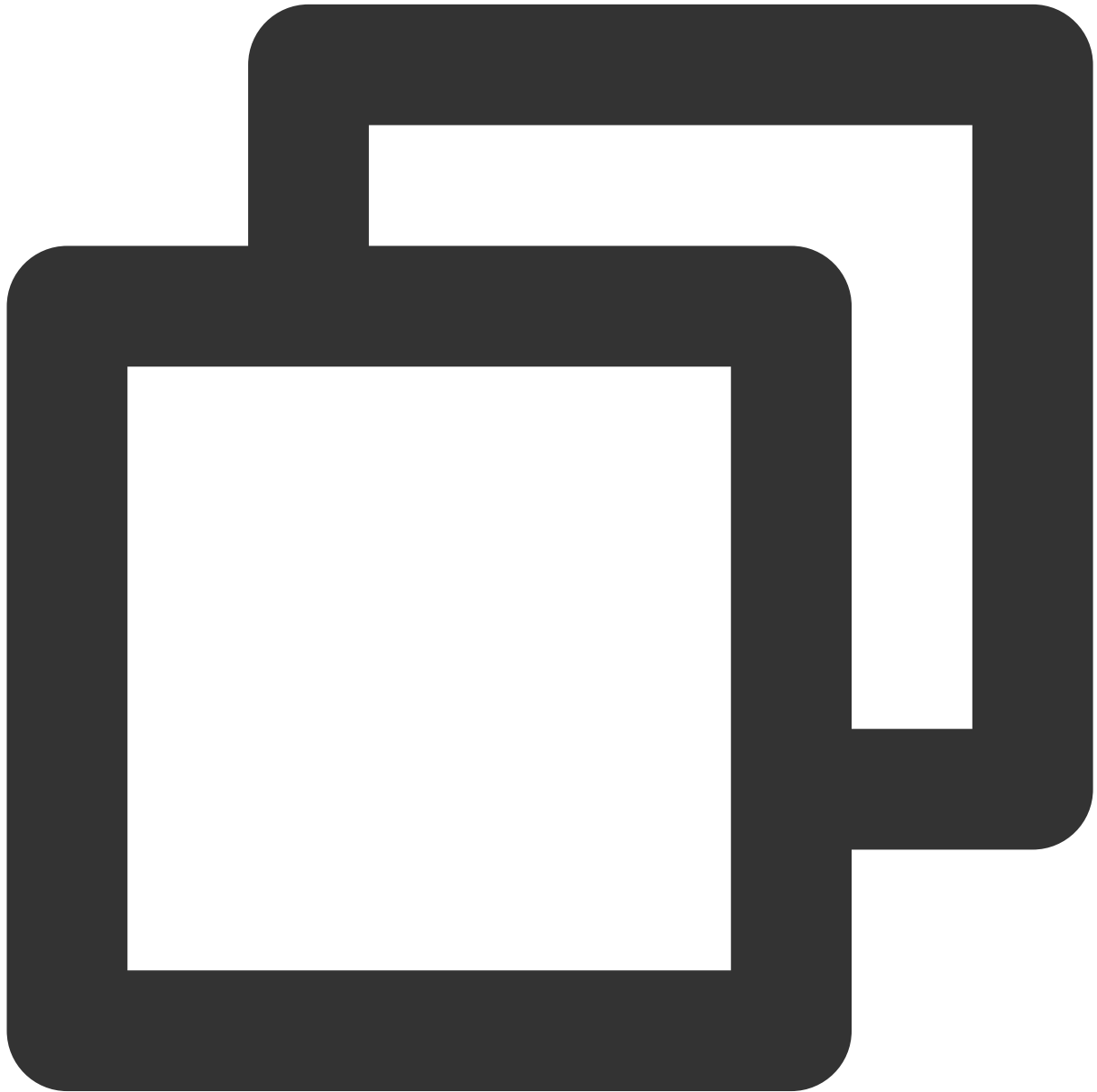


```
@ProxyService(proxy = MiniConfigProxy.class)
public class MiniConfigProxyImpl extends MiniConfigProxy {
    /**
     * 创建初始化配置信息
     * @return
     */
    @Override
    public MiniInitConfig buildConfig() {
        MiniInitConfig.Builder builder = new MiniInitConfig.Builder();
        MiniInitConfig config = builder
            .x5LicenseKey("")//设置LicenseKey
```

```
        .build();  
    }  
}
```

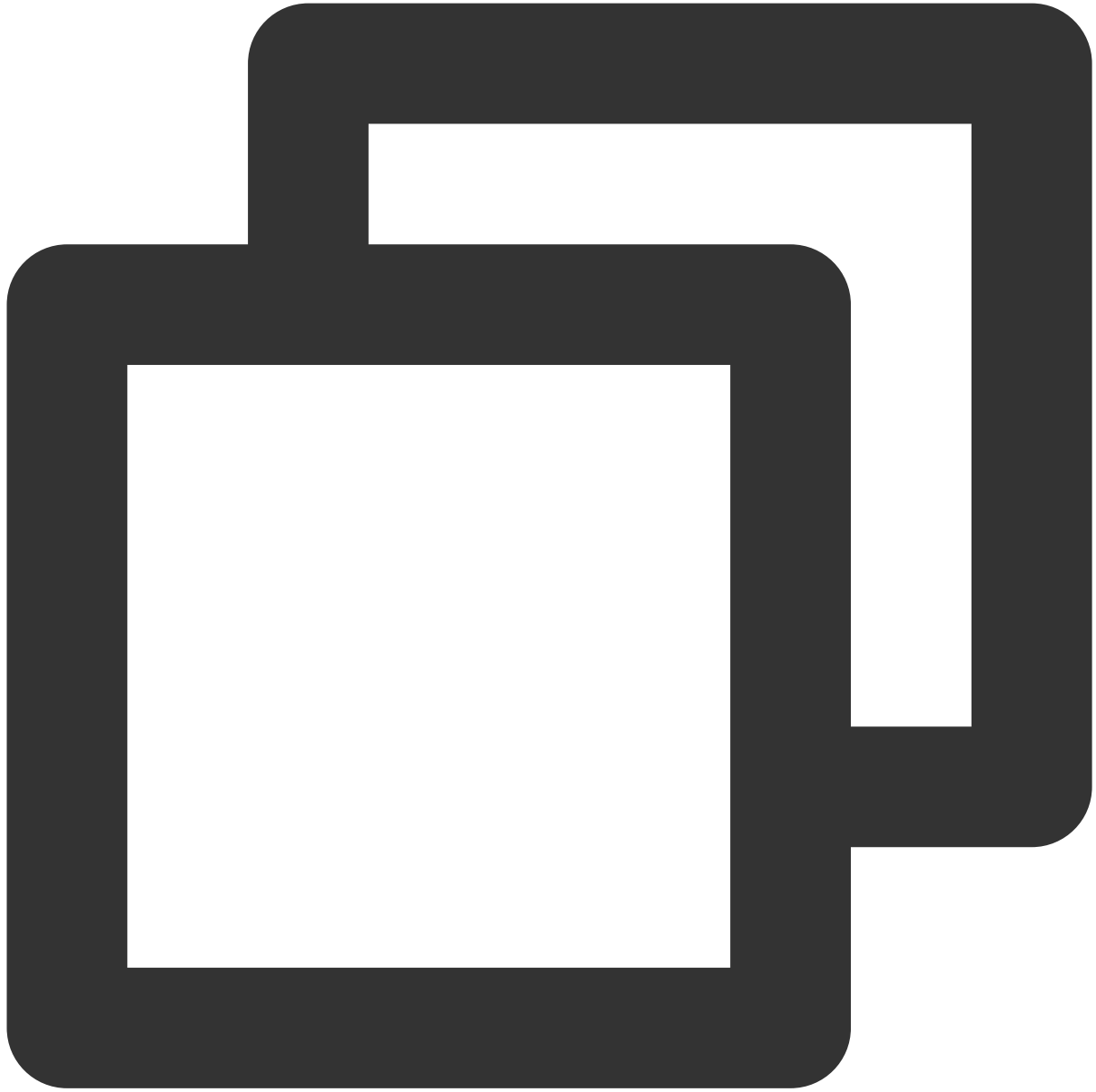
动态版 X5 内核

动态内核集成到 app 的 sdk 体积小，内核从服务端下载。



```
implementation 'com.tencent.tmf.android:dynamicx5:${version}'  
implementation 'com.tencent.tcmpp.android:mini_extra_dynamic_x5:${version}' //版本信
```

动态内核的开发者需要通过 `MiniInitConfig` 设置 `LicenseKey` 和内核下载地址，配置代码如下示例：

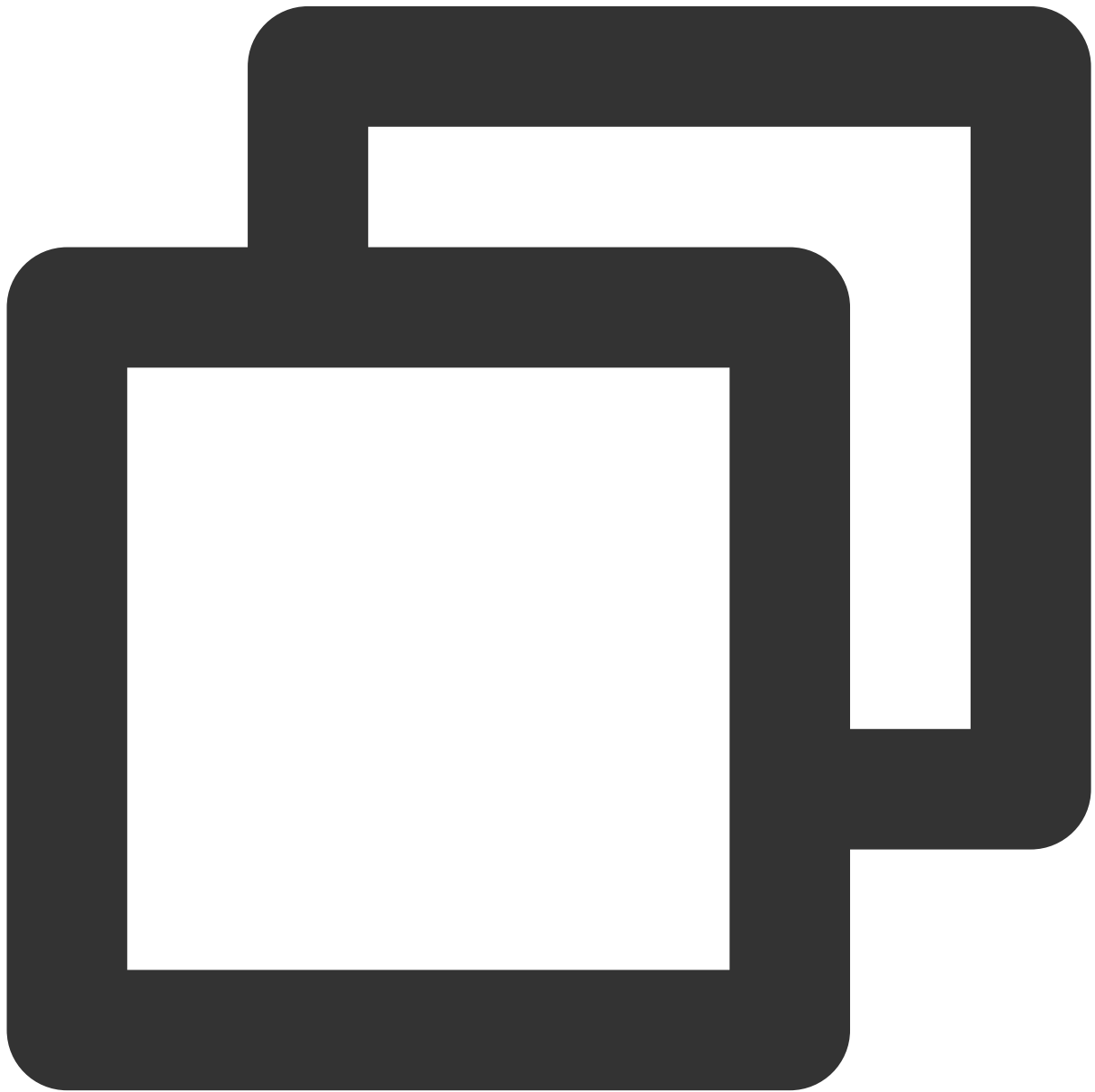


```
@ProxyService(proxy = MiniConfigProxy.class)
public class MiniConfigProxyImpl extends MiniConfigProxy {
    /**
     * 创建初始化配置信息
     * @return
     */
    @Override
    public MiniInitConfig buildConfig() {
        MiniInitConfig.Builder builder = new MiniInitConfig.Builder();
```

```
MiniInitConfig config = builder
    .x5LicenseKey("")//设置LicenseKey
    .coreUrl32("")//32位内核下载地址
    .coreUrl64("")//64位内核下载地址
    .build();
}
```

部署内核文件到服务器，并按照规则生成对应的 URL 地址。

内核文件 URL 地址的生成规则如下：



```
(http 或 https) ://domain/path/{versionCode}/{tbscore.tbs}
```

{versionCode}：为获取内核时提供的内核版本号。

{tbscore.tbs}：为内核文件。

需要保证 URL 以"/"截取时，内核文件为最后一位，versionCode 为倒数第二位。

示例如下：

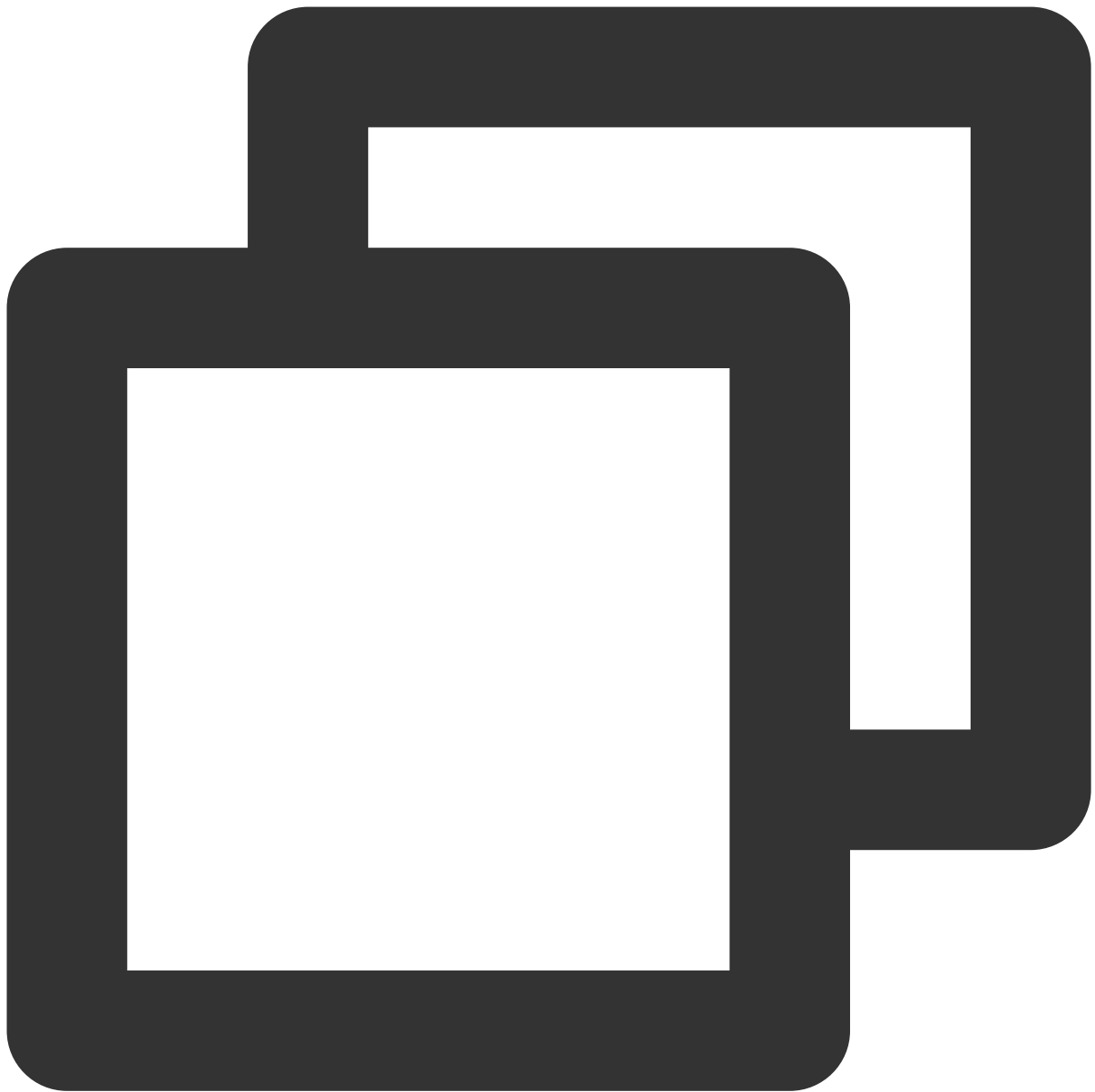
[32位内核下载地址](#)

[64位内核下载地址](#)

本例当中，内核版本号为 46471，内核文件为 tbs_core_release.tbs。

监听X5内核的事件

针对动态内核，可以通过如下的方式监听动态内核的下载和初始化结果：



```
@ProxyService(proxy = IX5EventProxy.class)
public class X5EventProxyImpl implements IX5EventProxy {
    /**
     * 动态内核下载进度
     * @param progress
     */
    @Override
    public void onDownloadProgress(int progress) {
        Log.d(ModuleApplet.TAG, "X5EventProxyImpl onDownloadProgress=" + progress);
    }
}
```

```
/**
 * 动态内核下载失败
 * @param code
 * @param msg
 */
@Override
public void onDownloadFailed(int code, String msg) {
    Log.d(ModuleApplet.TAG, "X5EventProxyImpl onDownloadFailed=" + code + " " + msg);
}

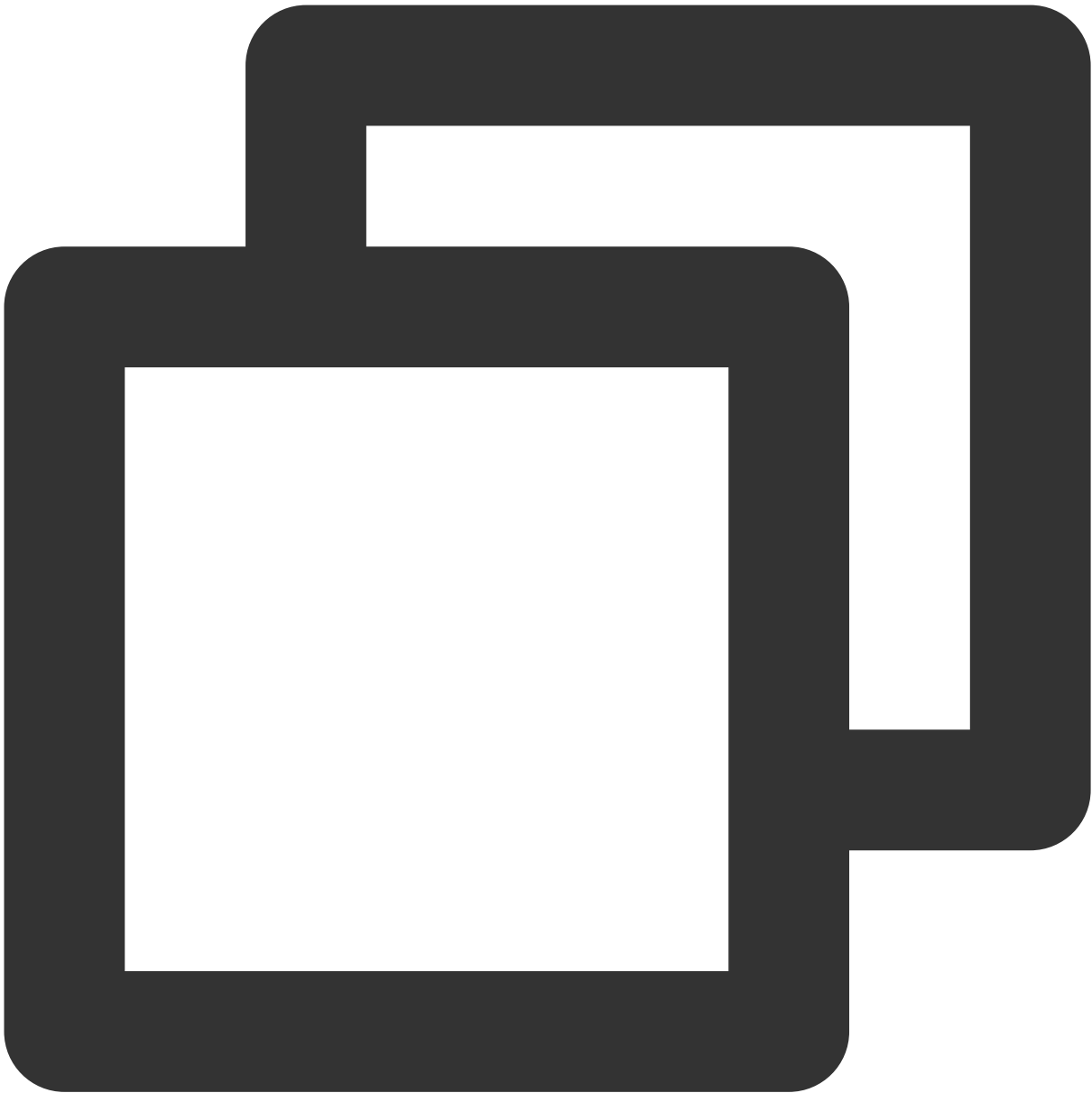
/**
 * 动态内核下载完成
 */
@Override
public void onDownloadFinish() {
    Log.d(ModuleApplet.TAG, "X5EventProxyImpl onDownloadFinish");
}

/**
 * 内核初始化回调
 * @param isX5 true:x5初始化成功;false:x5初始化失败
 */
@Override
public void init(boolean isX5) {
    Log.d(ModuleApplet.TAG, "X5EventProxyImpl isX5=" + isX5);
}
}
```

扫码扩展 SDK

组件说明：开发者小程序如果使用了小程序扫码能力，则需要添加如下 SDK 支持扫码功能；如未使用，建议暂不添加，这样可以减小 App 包大小。

集成方法：按照如下的方式添加扫码扩展库依赖：



```
//扫码扩展组件
implementation 'com.tencent.tcmpp.android:mini_extra_qrcode:${version}' // 版本信息请
```

添加扫码扩展 SDK 后，增加支持的小程序 API 列表如下：

API 名称	说明
wx.scanCode	调起客户端扫码界面进行扫码

涉及权限：

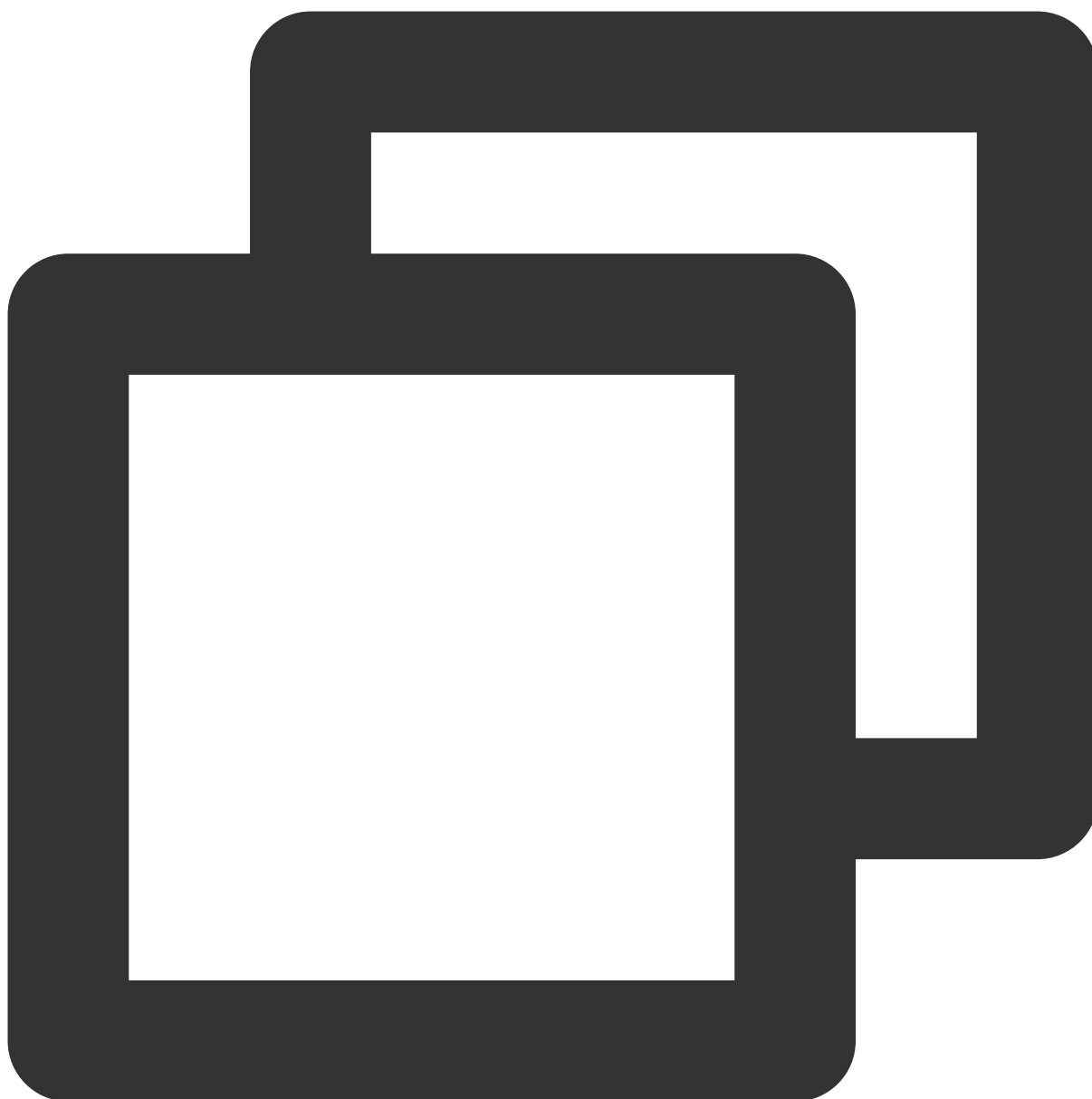
--	--

权限	描述
相机权限	需要申请相机权限用于扫码
文件读写权限	需要申请文件读写权限用于识别本地图片中的二维码

腾讯定位地图扩展 SDK

组件说明：针对中国大陆地区 App 开发，开发者小程序如果使用了小程序地图能力，则需要添加如下 SDK 支持腾讯地图功能；如未使用，建议暂不添加，这样可以减小 App 包大小。

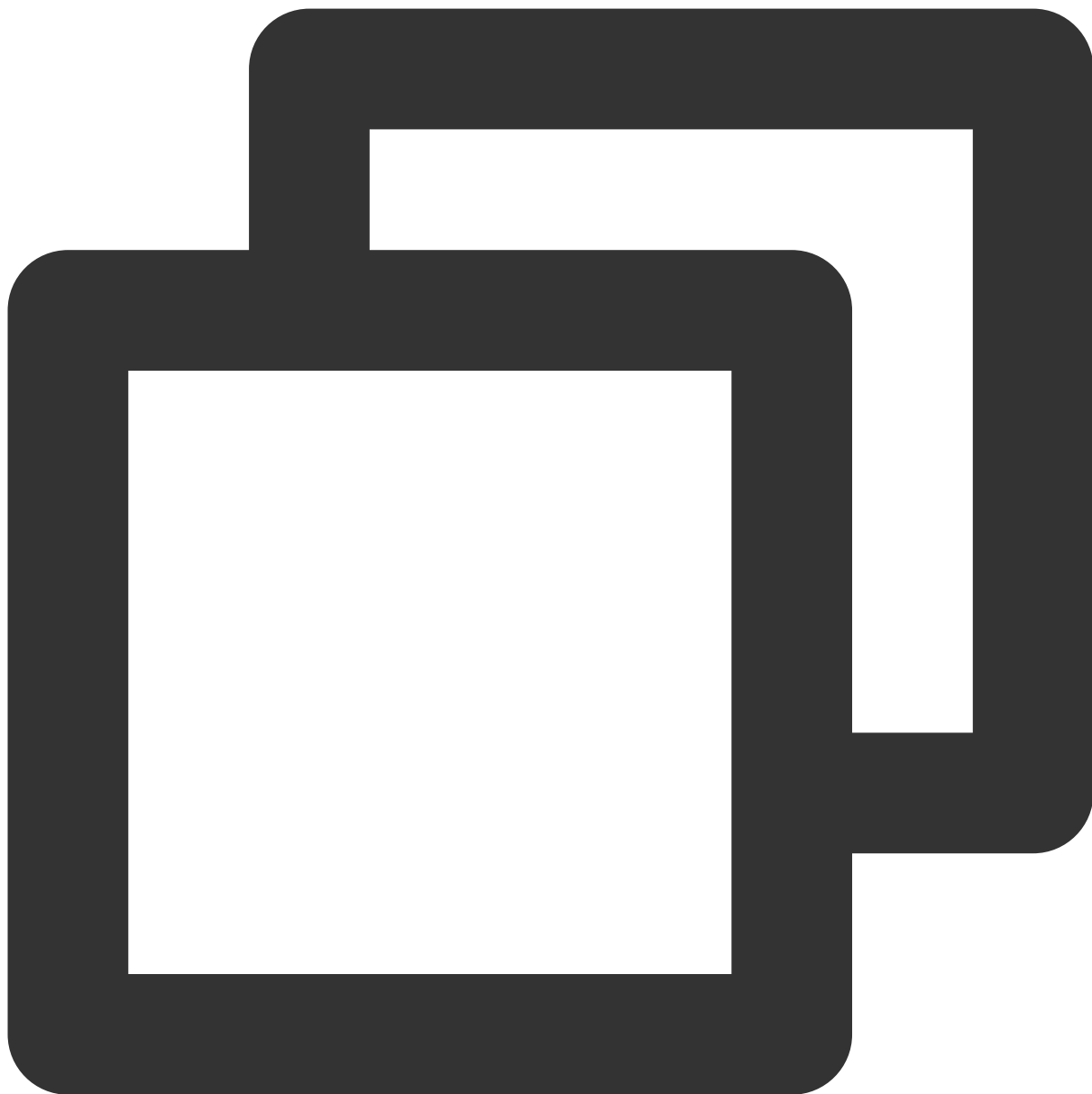
集成方法：按照如下的方式添加地图扩展库依赖：



```
implementation 'com.tencent.tcmpp.android:mini_extra_map:${version}'// 版本信息请参考  
implementation 'com.tencent.map:tencent-map-vector-sdk:4.5.10' //版本信息参考腾讯地图  
implementation 'com.tencent.map:sdk-utilities:1.0.7'  
implementation 'com.tencent.map.geolocation:TencentLocationSdk-openplatform:7.4.7'
```

您需要在您的腾讯位置服务控制台配置项目，并获取访问腾讯地图服务所需要的 API 密钥，详细操作，请参考 [开发指南](#)。

完成上述操作后，您需要在 Android 工程中配置您的 API 密钥。在 AndroidManifest.xml 文件中添加以下 meta-data，并将您的 API 密钥填入 (YOUR_API_KEY) 位置：



```
<application
  ...
  <meta-data
    android:name="TencentMapSDK"
    android:value="(YOUR_API_KEY)" />
  ...
</application>
```

添加腾讯地图扩展 SDK 后，增加支持的小程序 API 列表如下：

API 名称	说明
--------	----

地图	支持地图相关接口，包括地图展示，使用地图选择位置以及查询 POI 等
----	------------------------------------

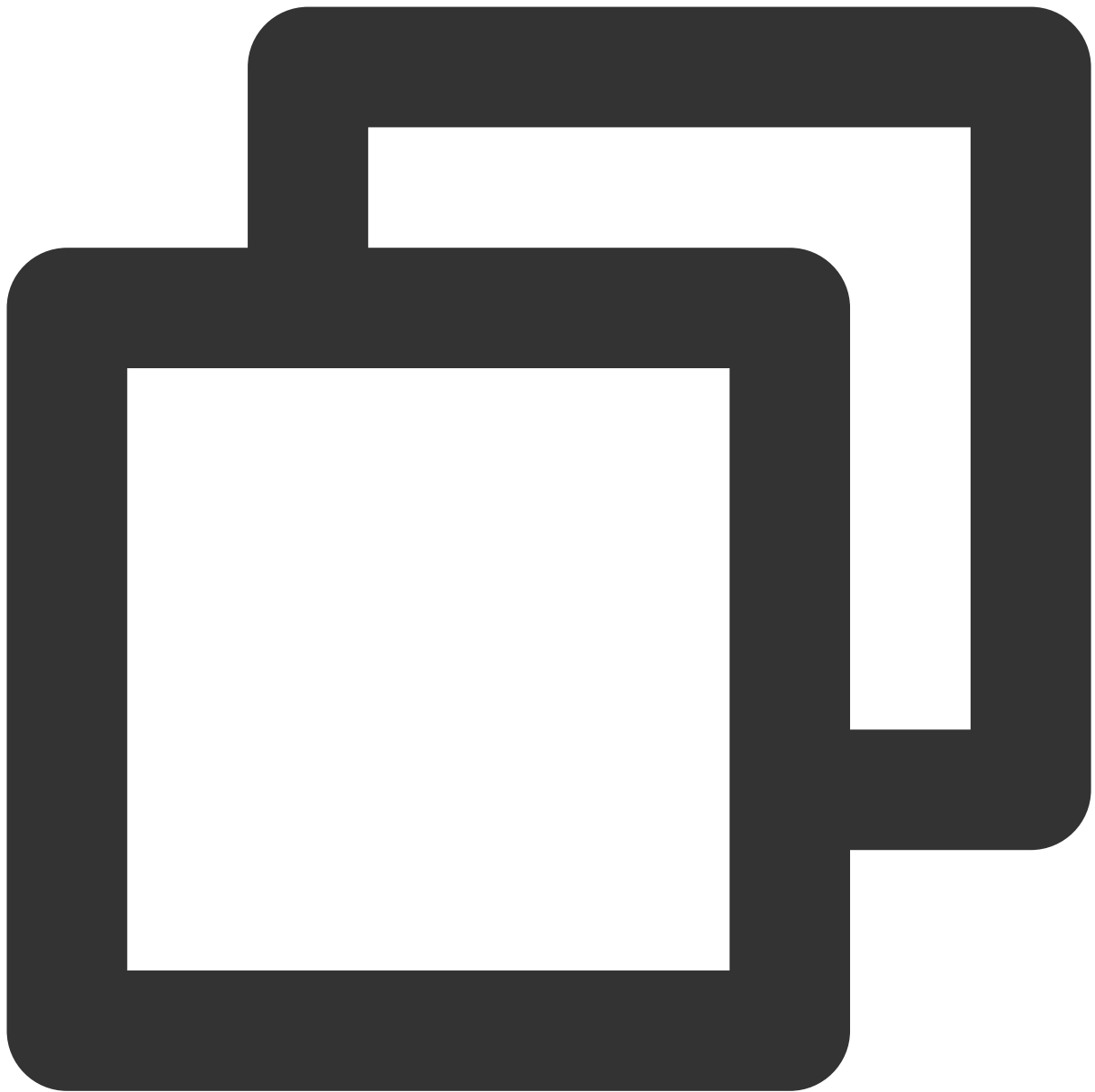
涉及权限：

权限	描述
定位权限	需要使用定位权限用于显示地图定位

Google 及华为定位地图扩展 SDK

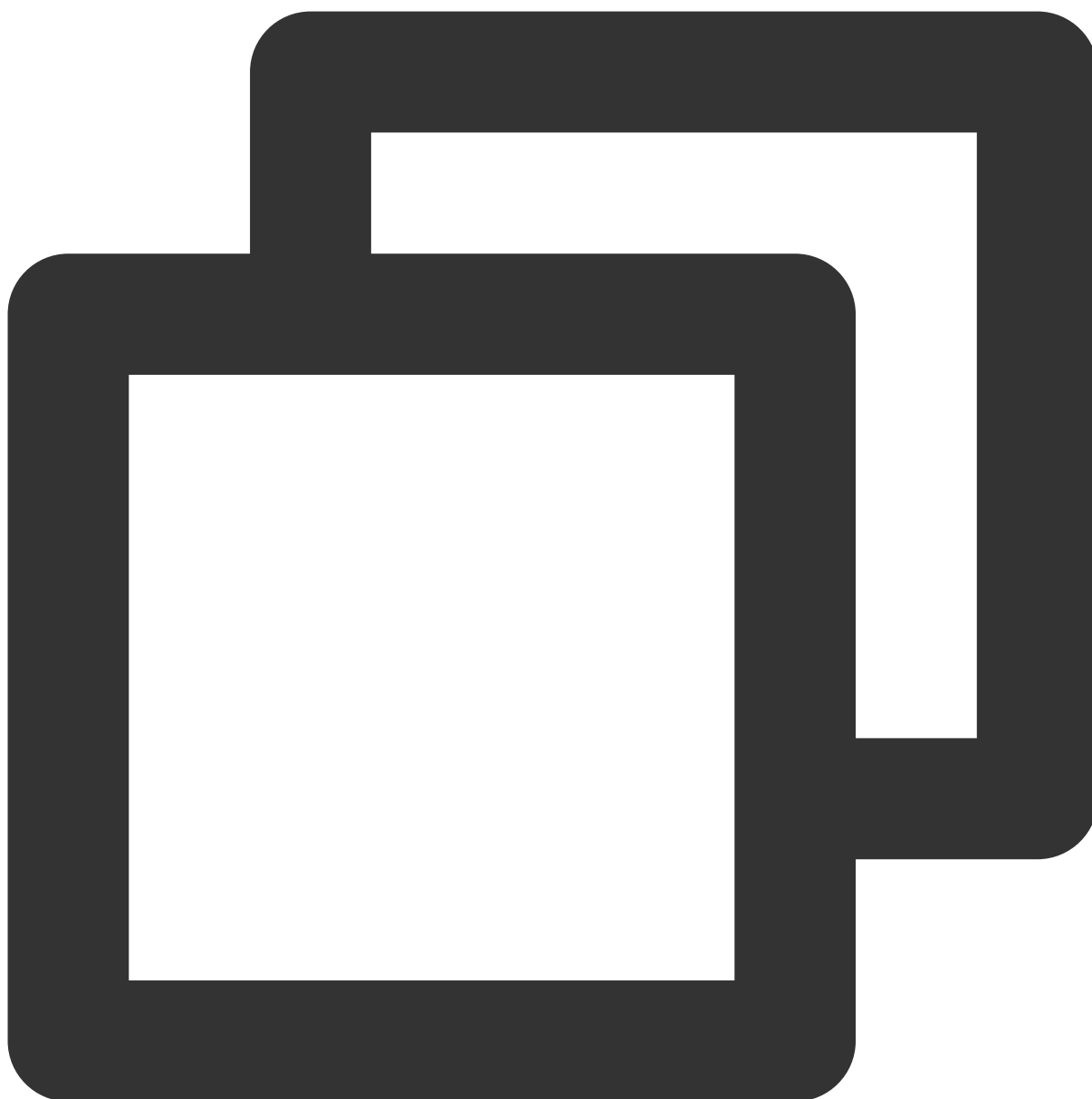
组件说明：针对境外 App 开发，开发者小程序如果使用了小程序地图能力，需要添加如下 SDK 支持 Google Map 功能；如未使用，建议暂不添加，这样可以减小 App 包大小。

集成方法：按照如下的方式添加地图扩展库依赖：



```
implementation 'com.tencent.tcmpp.android:mini_extra_google_map:${version}'//版本信息
implementation 'com.google.android.gms:play-services-maps:18.1.0'//版本信息
implementation 'com.google.maps.android:android-maps-utils:2.3.0'
```

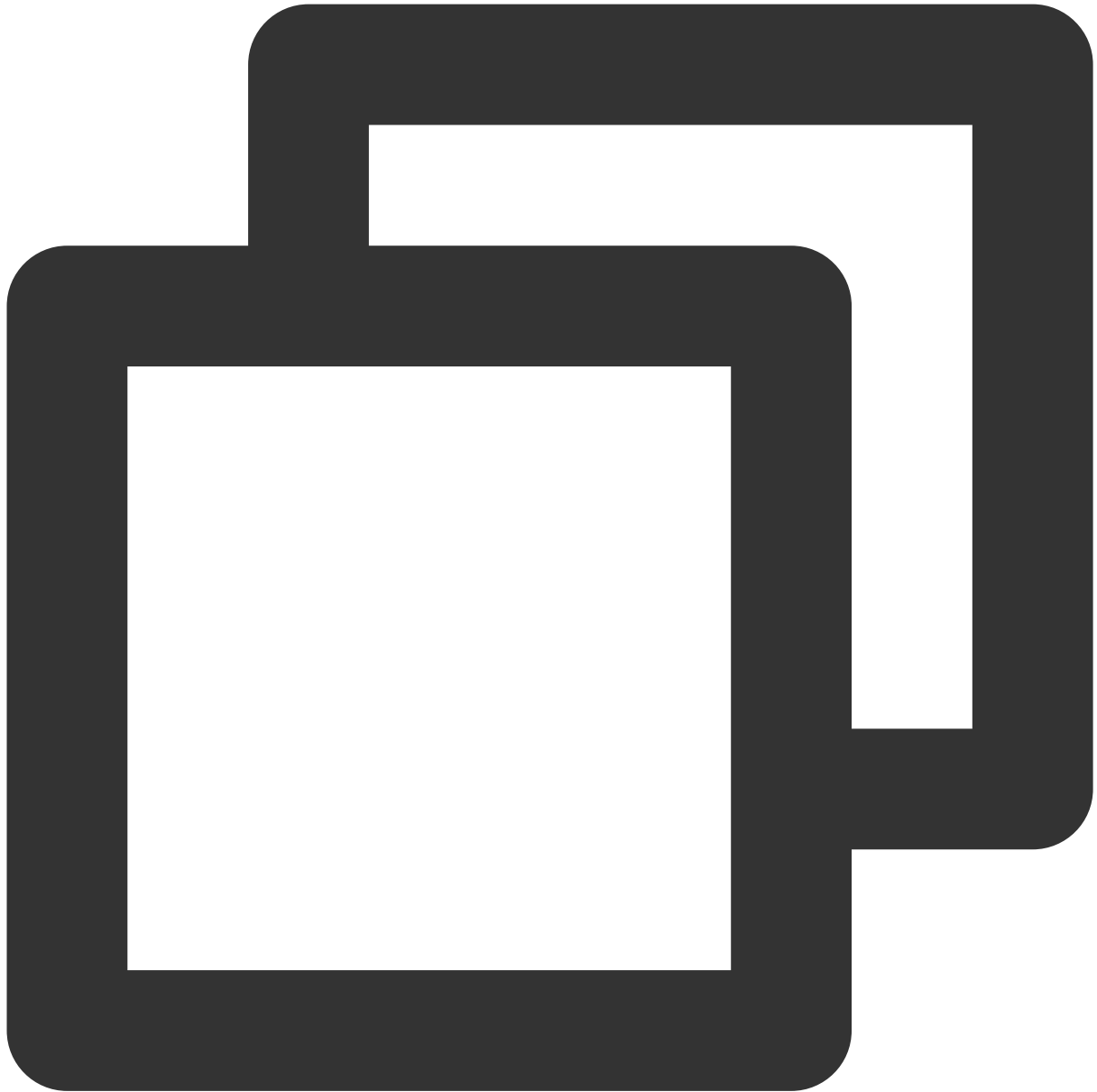
由于部分华为设备不支持内嵌 Google Map，可能导致地图无法显示。您可以额外接入 Petal Map 作为补充方案，小程序框架将在华为设备上优先使用 Petal Map。



```
repositories {  
    maven {url 'https://developer.huawei.com/repo/'}  
}  
implementation 'com.tencent.tcmpp.android:mini_extra_huawei_map:${version}'//版本信息  
implementation 'com.huawei.hms:maps:6.9.0.300'//版本信息  
implementation 'com.huawei.hms:maps-basic:6.9.0.300'  
implementation 'com.huawei.hms:site:6.5.1.300'
```

使用 Google Map 的情形，您需要在您的 Google Cloud Console 配置 Google Cloud 项目，并获取访问 Google 地图服务所需要的 API 密钥，具体操作步骤请参考在 [Google Cloud Console 中进行设置](#) 以及使用 [API 密钥](#)。

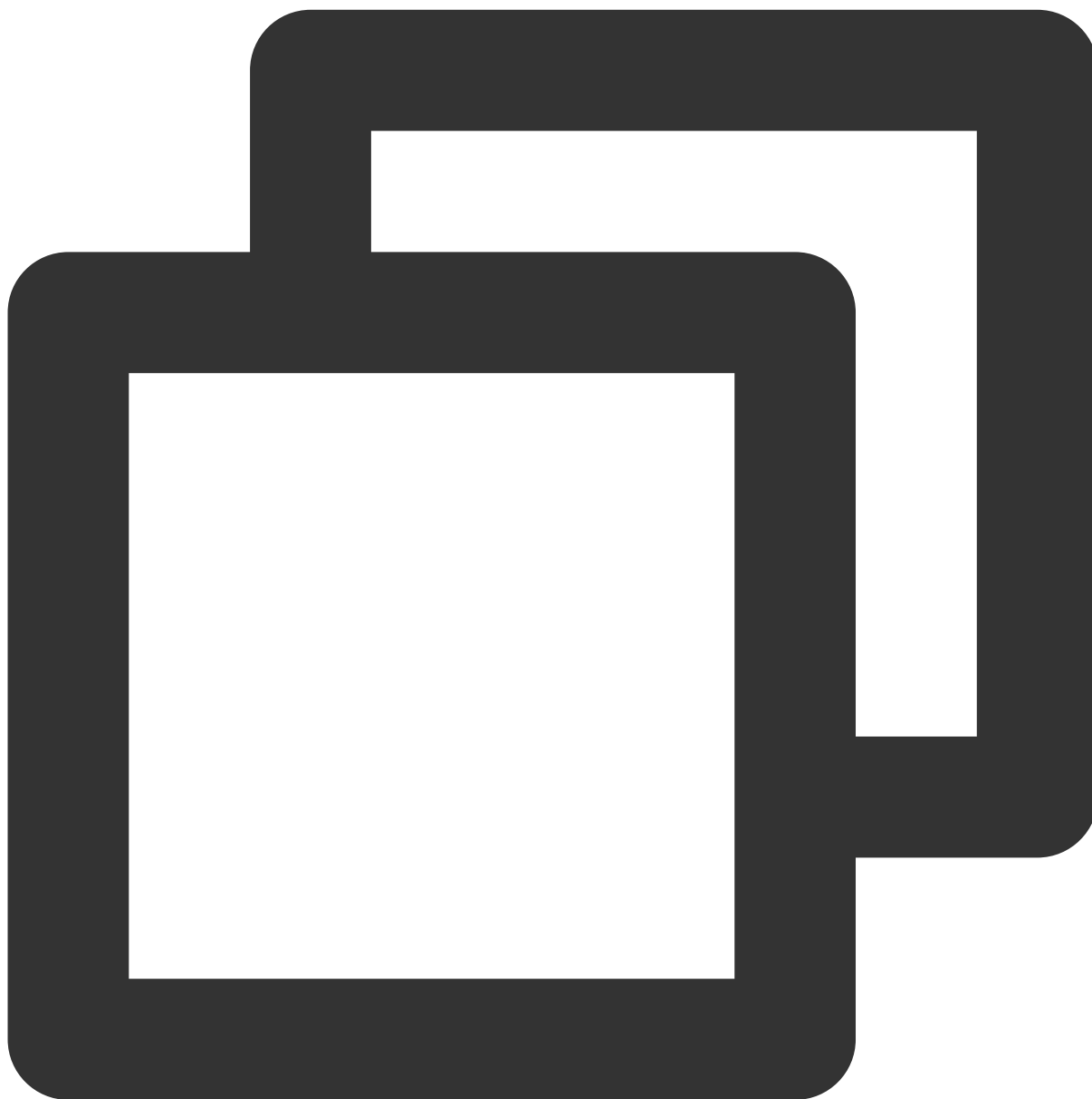
完成上述操作后，您需要在 Android 工程中配置您的 API 密钥。在 AndroidManifest.xml 文件中添加以下 meta-data，并将您的 API 密钥填入 (YOUR_API_KEY) 位置：



```
<application
  ...
  <meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value="(YOUR_API_KEY)" />
  ...
</application>
```

使用 Petal Map 的情形，您需要在您的华为管理控制台建立项目、开通地图以及位置服务并获取位置服务所使用的 API 密钥，具体操作步骤，请参考 [配置 AppGallery Connect](#)。然后按照 [集成 HMS Core SDK](#) 的引导下载“agconnect-services.json”文件至您的项目中并配置华为 AGC 插件。

您需要在 AndroidManifest.xml 文件中添加以下 meta-data，并将您的 API 密钥填入 (YOUR_API_KEY) 位置以正常使用华为的位置服务：



```
<application
...
<meta-data
    android:name="HuaweiApiKey"
```

```
    android:value="(YOUR_API_KEY)" />
    ...
</application>
```

注意：

出于安全考虑，建议您为位置服务单独申请 API 密钥。
添加 Google、华为地图扩展 SDK 后，增加支持的小程序 API 列表如下：

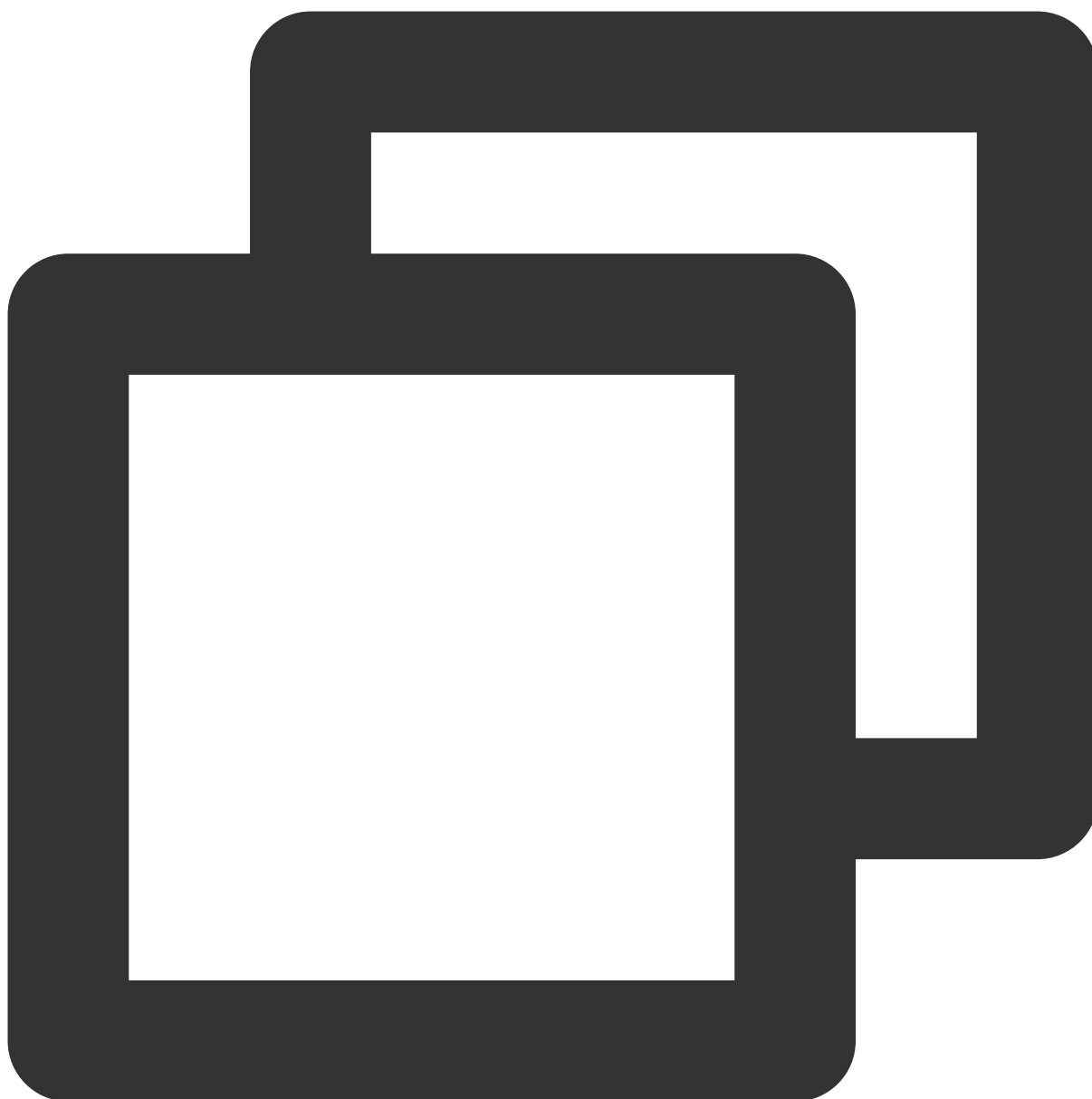
API 名称	说明
地图	支持地图相关接口以及组件，包括地图展示，使用地图选择位置以及查询 POI 等

涉及权限：

权限	描述
定位权限	需要使用定位权限用于显示地图定位

直播组件扩展 SDK

组件说明：如果您需要使用直播组件（live-player 和 live-pusher）进行直播推、拉流相关场景的开发，需要添加如下 SDK 以支持直播组件相关的功能的实现。
集成方法：添加直播组件依赖。

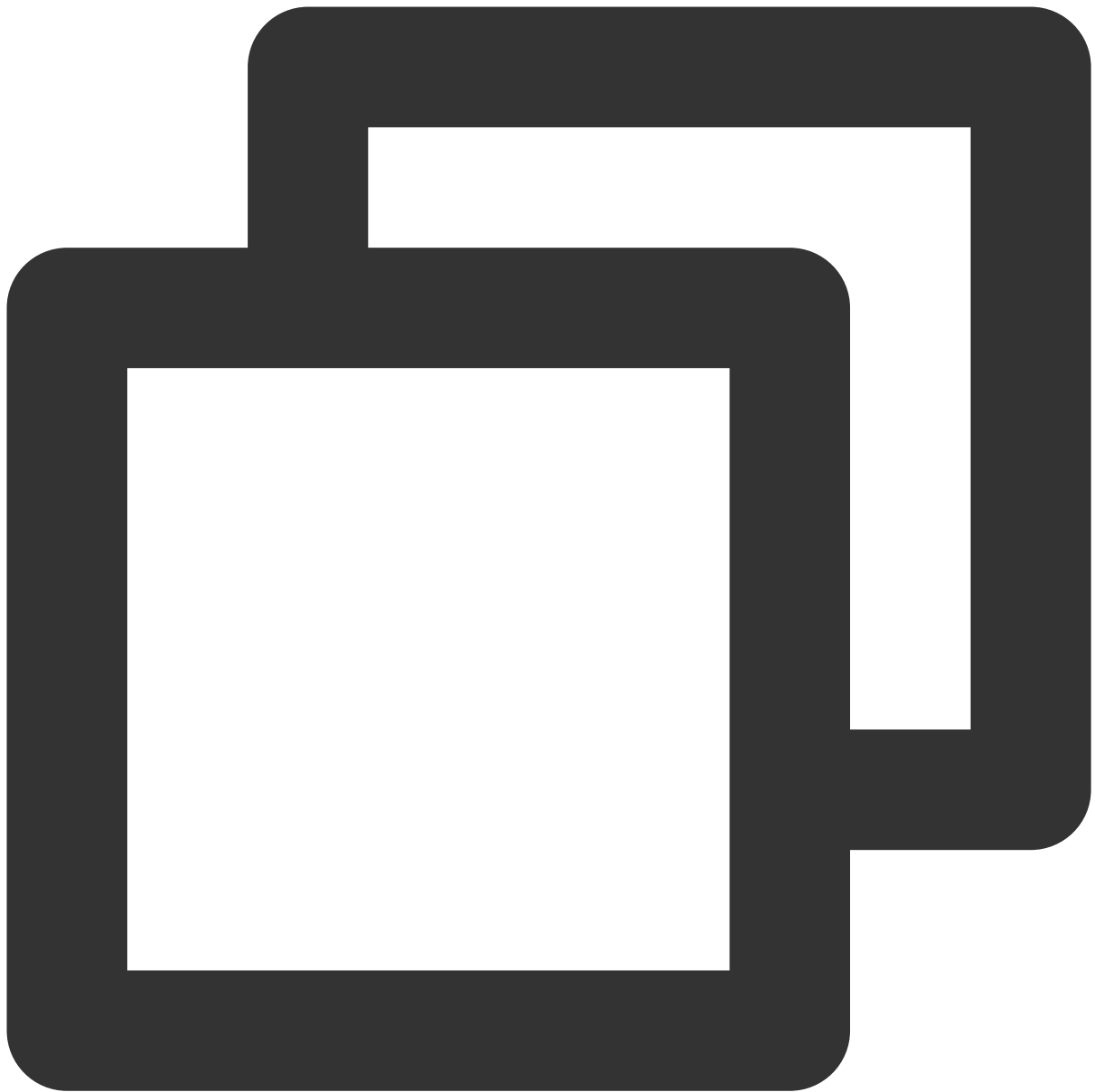


```
//直播组件支持库  
implementation 'com.tencent.tcmpp.android:mini_extra_trtc_live:${version}'//版本信息  
//直播组件库  
implementation 'com.tencent.liteav:LiteAVSDK_Professional:latest.release' //版本信息
```

除了完成以上依赖的添加，您还需要重写实现 `BaseMiniAppProxyImpl` 的如下方法，提供直播组件需要的 `LicenseUrl` 和 `LicenseKey`，以完成直播组件的初始化信息配置；如果您未配置正确的 `LicenseUrl` 和 `LicenseKey`，会导致直播组件功能不可用。

说明：

`LicenseUrl` 和 `LicenseKey` 的获取方式可参考 [新增与续期 License](#)。



```
@ProxyService(proxy = MiniAppProxy.class)
public class MiniAppProxyImpl extends BaseMiniAppProxyImpl {
    @Override
    public MiniConfigData configData(Context context, int configType, JSONObject pa
        if(configType == MiniConfigData.TYPE_LIVE) {
            //Live直播配置
            MiniConfigData.LiveConfig liveConfig = new MiniConfigData.LiveConfig();
            //下面的key和url仅可用于demo
            liveConfig.licenseKey = "";
            liveConfig.licenseUrl = "";
        }
    }
}
```

```
        return new MiniConfigData
            .Builder()
            .liveConfig(liveConfig)
            .build();
    }

    return null;
}
```

添加直播扩展 SDK 后，增加支持的小程序 API 列表如下：

API 名称	说明
wx.createLivePusherContext	创建直播推流端 context
LivePusherContext	支持 LivePusherContext 相关接口
wx.createLivePlayerContext	创建直播拉流端 context
LivePlayerContext	支持 LivePlayerContext 相关接口
Component	-
live-pusher	推流标签
live-player	播放标签

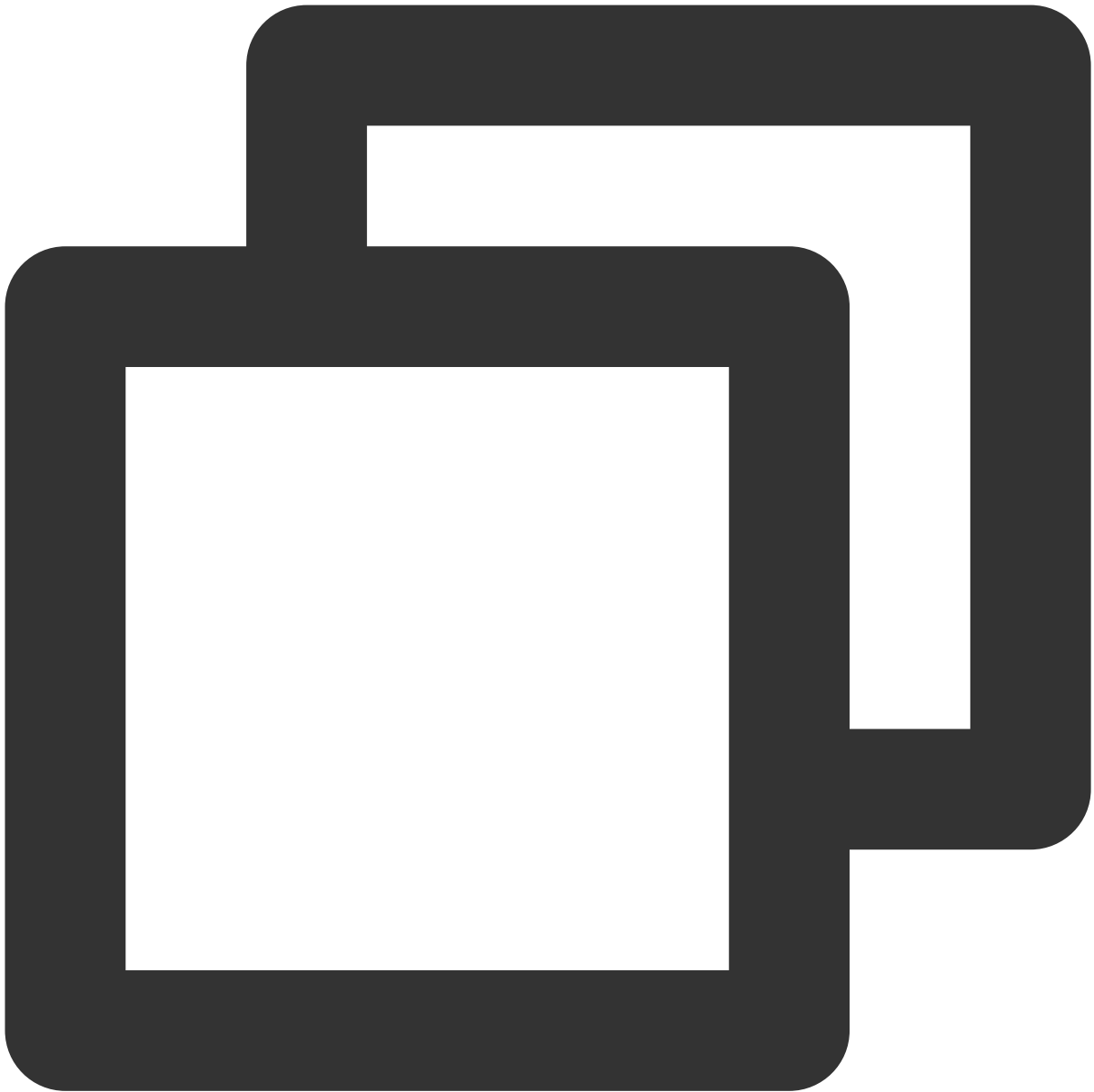
涉及权限如下：

权限名称	描述
相机权限	-
录音权限	-

LBS 扩展 SDK

组件说明：LBS 组件提供位置信息、罗盘、加速计、定位、设备方向的相关的能力。

集成方法：按照如下的方式添加 LBS 扩展库依赖：



```
implementation 'com.tencent.tcmpp.android:mini_extra_lbs:${version}'//版本信息请参考A1
```

添加 LBS 扩展 SDK 后，增加支持的小程序 API 列表如下：

API 名称	说明
位置信息	支持位置信息相关接口
罗盘	支持罗盘相关接口
加速计	支持加速计相关接口

设备方向	支持设备方向相关接口
陀螺仪	支持陀螺仪相关接口

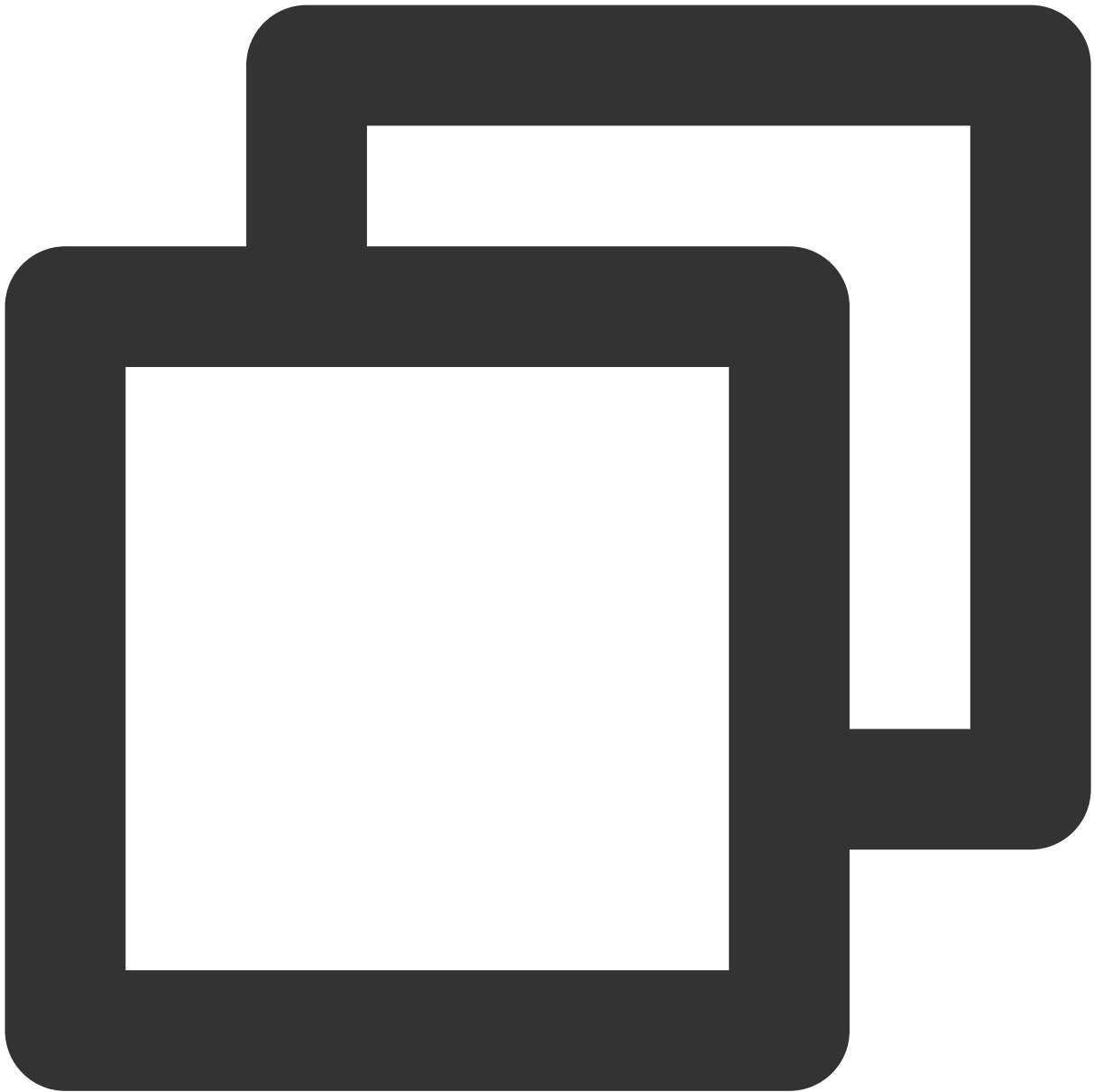
LBS 扩展 SDK 涉及权限如下：

权限	说明
定位	获取定位依赖定位权限

蓝牙扩展 SDK

组件说明：添加蓝牙扩展库之后，即可使用蓝牙相关的 API。

集成方法：按照如下的方式添加蓝牙扩展库依赖：



```
implementation 'com.tencent.tcmpp.android:mini_extra_bluetooth:${version}'//版本信息
```

添加 LBS 扩展 SDK 后，增加支持的小程序 API 列表如下：

API	说明
蓝牙-通用	蓝牙通用接口
蓝牙-低功耗外围设备	外围设备相关接口
蓝牙-低功耗中心设备	中心设备相关接口

蓝牙-信标	蓝牙信标相关接口
-------	----------

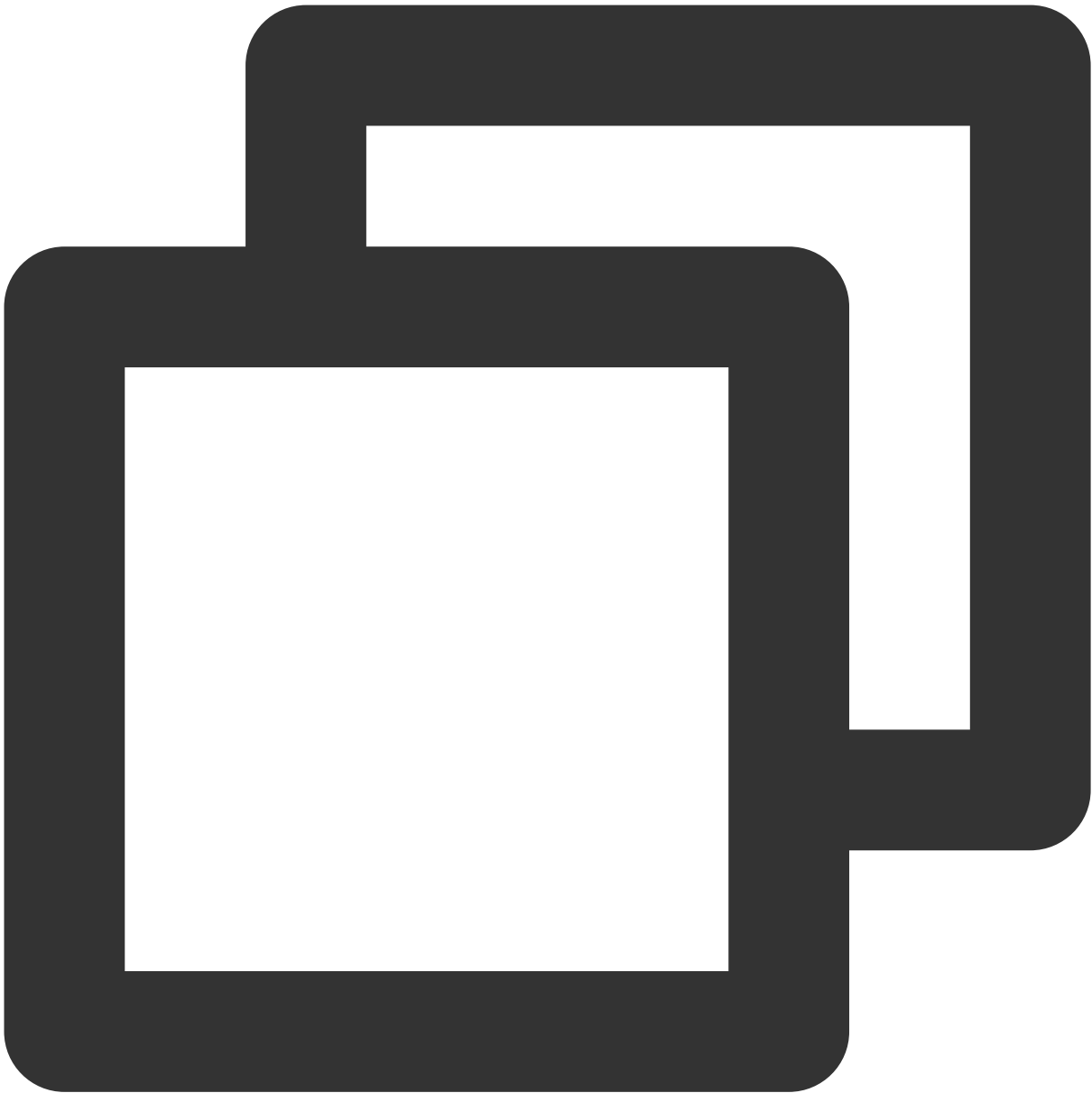
蓝牙扩展 SDK 涉及权限如下：

权限	说明
蓝牙	操作蓝牙需要申请蓝牙权限
定位	蓝牙设备搜索依赖定位权限

NFC 扩展 SDK

组件说明：添加 NFC 扩展 SDK，能够实现 NFC 读写相关的能力。

集成方法：按照如下的方式添加 NFC 扩展 SDK 依赖：



```
implementation 'com.tencent.tcmpp.android:mini_extra_nfc:${version}'//版本信息请参考A1
```

添加 NFC 扩展 SDK 后，增加支持的小程序 API 列表如下：

API 名称	说明
wx.getNFCAdapter	获取 NFC 操作管理实例对象
NFCAdapter	支持 NFCAdapter 相关接口
NFC 实例（NfcA、NfcB、NfcV、NfcF、Ndef、	支持 NFC 标签实例相关接口

IsoDep、MifareUltralight、MifareClassic)	
--	--

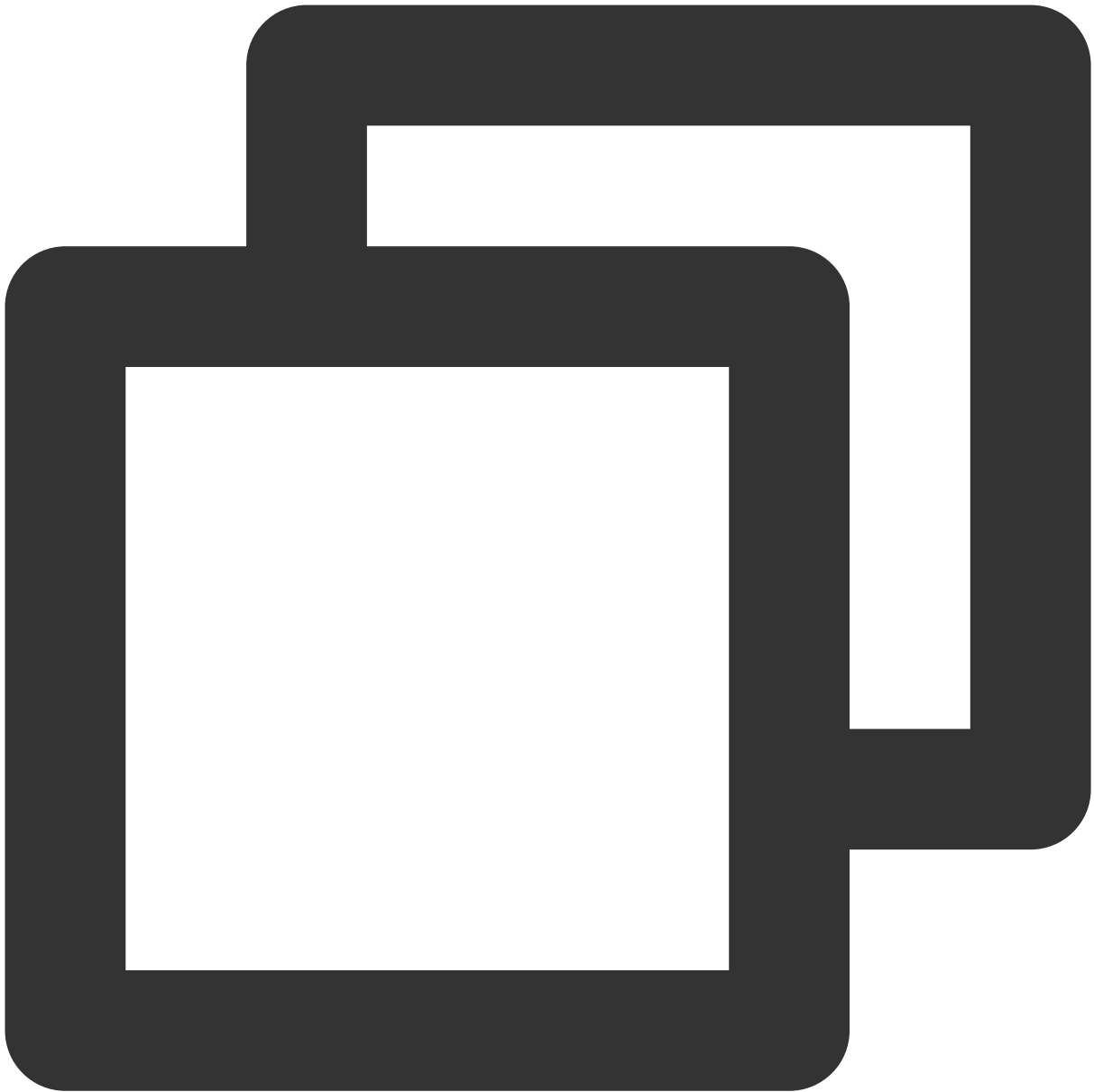
所涉及权限：

权限名称	描述
NFC 权限	需要获取 NFC 权限

生物认证扩展 SDK

组件说明：生物认证扩展 SDK 提供设备指纹、人脸识别相关的能力。

集成方法：按照如下的方式添加生物认证扩展库依赖：



```
implementation 'com.tencent.tcmpp.android:mini_extra_soter:${version}'//版本信息请参考
```

添加生物认证扩展 SDK 后，增加支持的小程序 API 列表如下：

API名称	说明
wx.startSoterAuthentication	-
wx.checkIsSupportSoterAuthentication	-
wx.checkIsSoterEnrolledInDevice	-

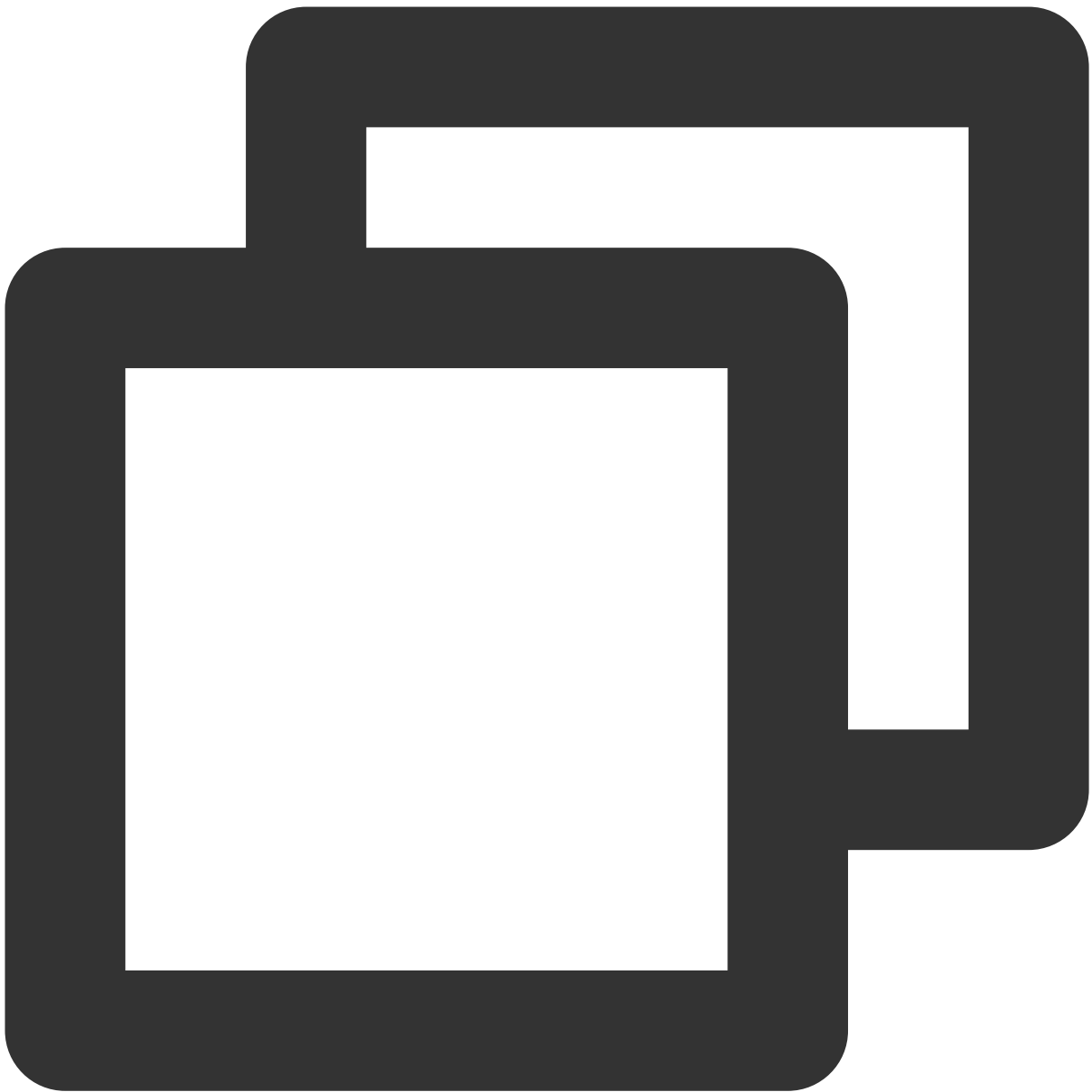
涉及权限：

权限	描述
指纹访问	需要申请指纹访问权限

剪切板扩展 SDK

组件说明：提供剪切板访问的能力。

集成方法：按照如下的方式添加扩展库依赖：



```
implementation 'com.tencent.tcmpp.android:mini_extra_clipboard:${version}'//版本信息
```

添加 LBS 扩展 SDK 后，增加支持的小程序 API 列表如下：

API名称	说明
wx.getClipboardData	-
wx.setClipboardData	-

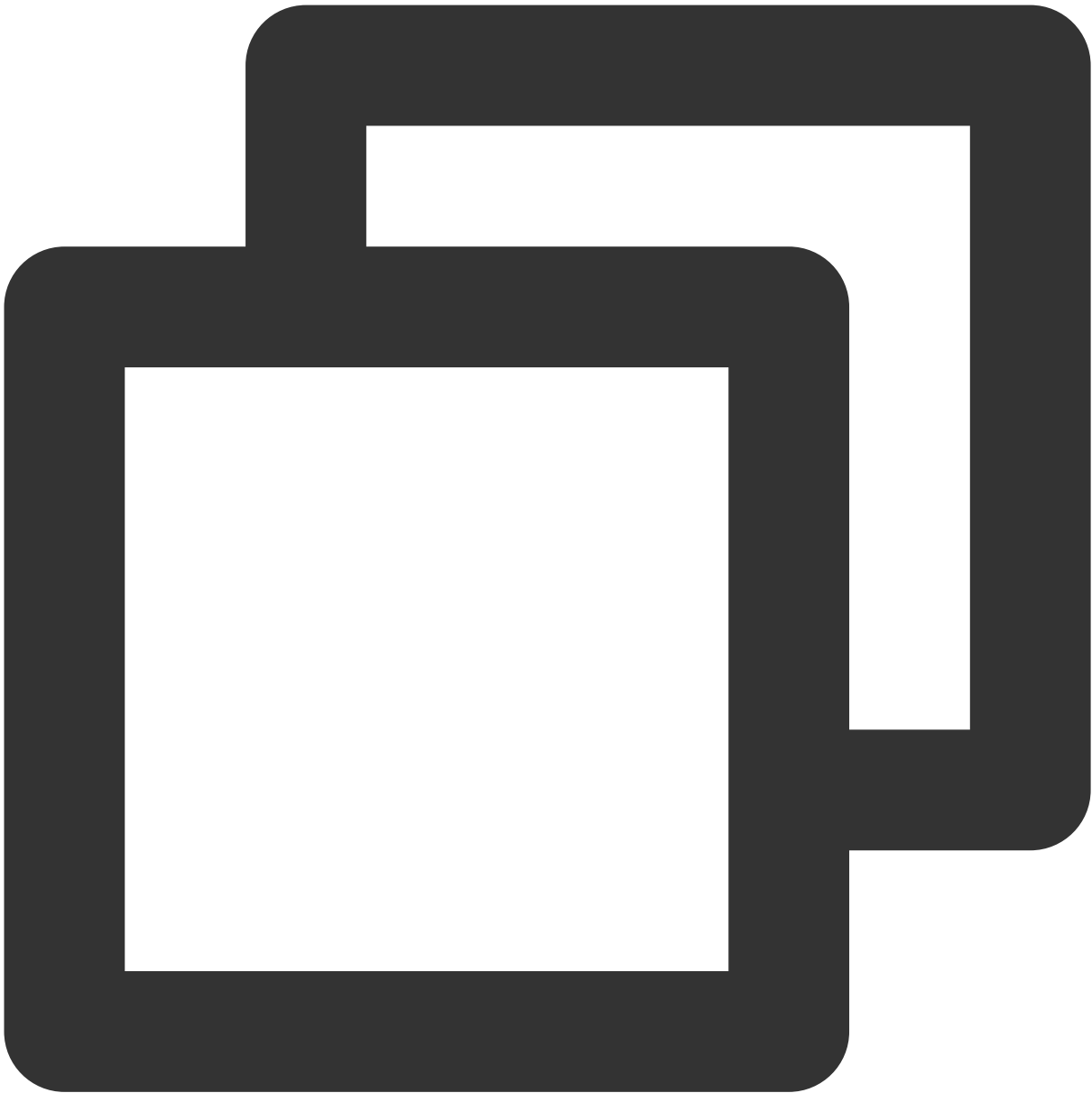
涉及权限：

权限	描述
剪贴板权限	需要申请剪切板访问权限

通讯录扩展 SDK

组件说明：提供联系人访问相关能力。

集成方法：按照如下的方式添加扩展库依赖：



```
implementation 'com.tencent.tcmpp.android:mini_extra_contact:${version}'//版本信息请参考
```

添加通讯录扩展 SDK 后，增加支持的小程序 API 列表如下：

API 名称	说明
wx.addPhoneContact	添加联系人
wx.chooseContact	选择联系人

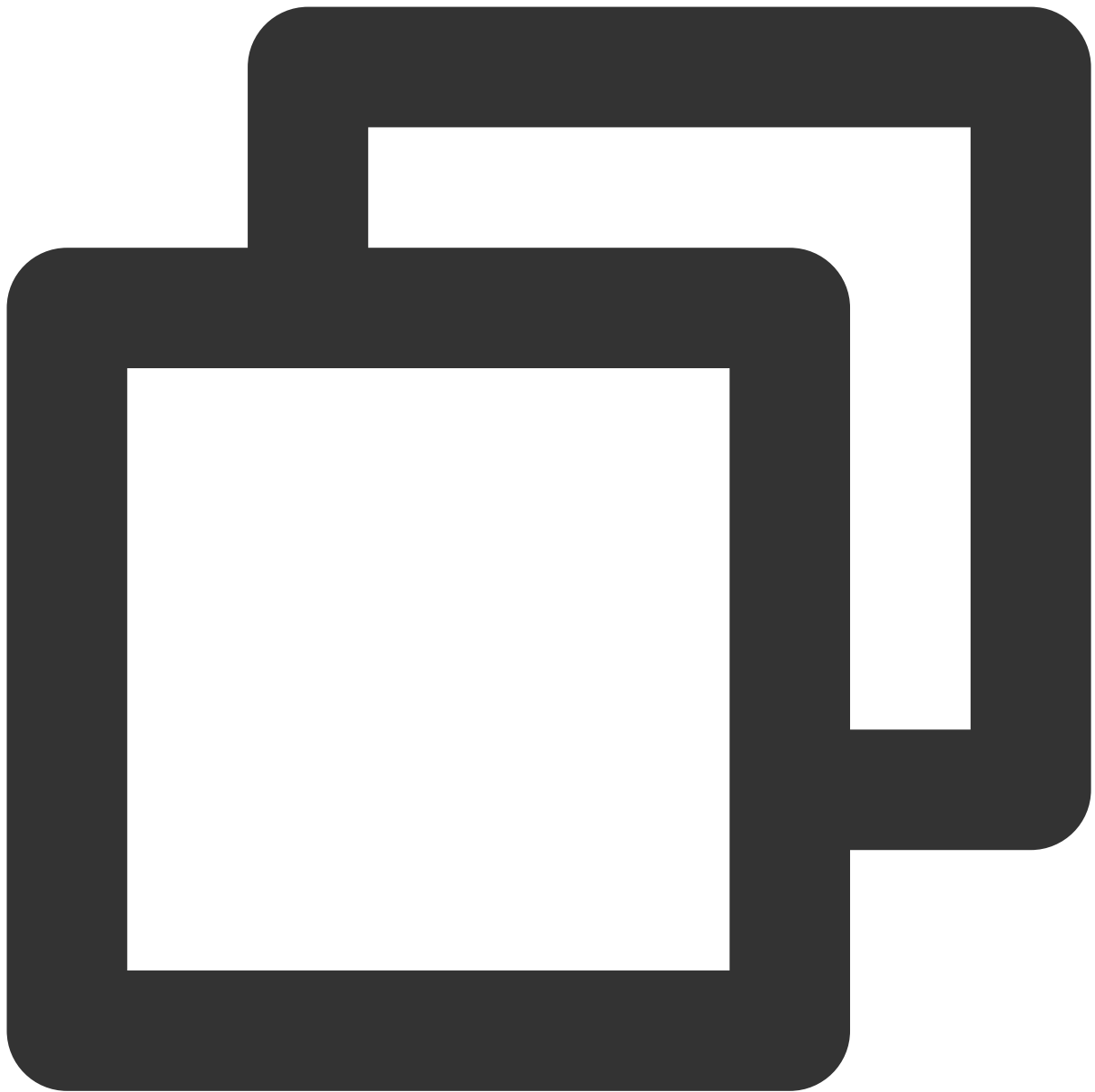
涉及权限：

权限	描述
联系人读写权限	需要申请联系人访问、写入权限

文档引擎扩展 SDK

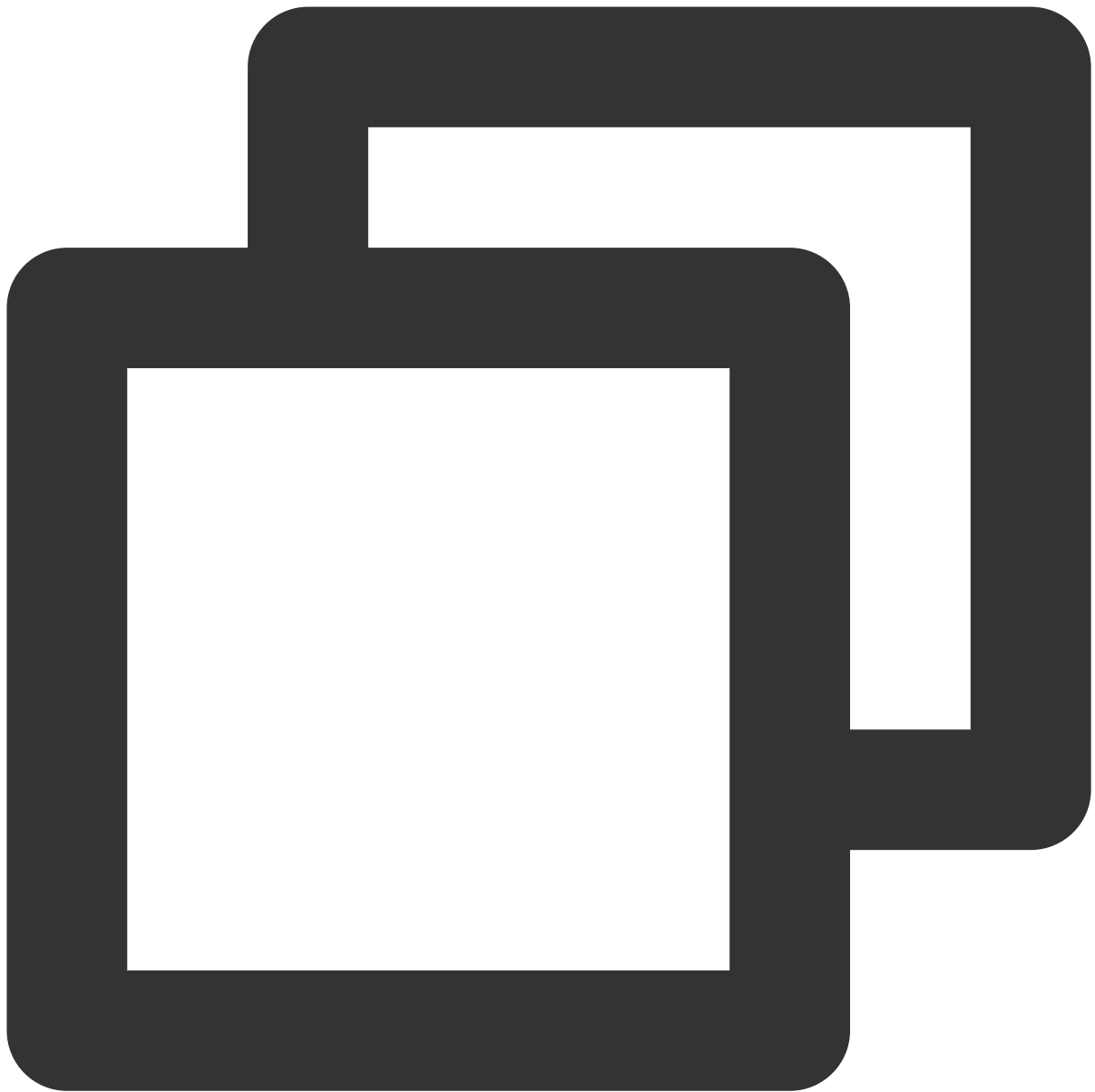
组件说明：提供文档（pdf、word、excel、ppt等）打开能力。

集成方法：按照如下的方式添加扩展库依赖：



```
implementation 'com.tencent.tmf.android:tbs-doc-support:${version}'//版本信息请参考Android  
implementation 'com.tencent.tcmpp.android:mini_extra_doc:${version}'//版本信息请参考 Android
```

配置文档引擎 LicenseKey，配置代码详细说明参考 [SDK 初始化](#)，LicenseKey 需要找相关人员进行申请；



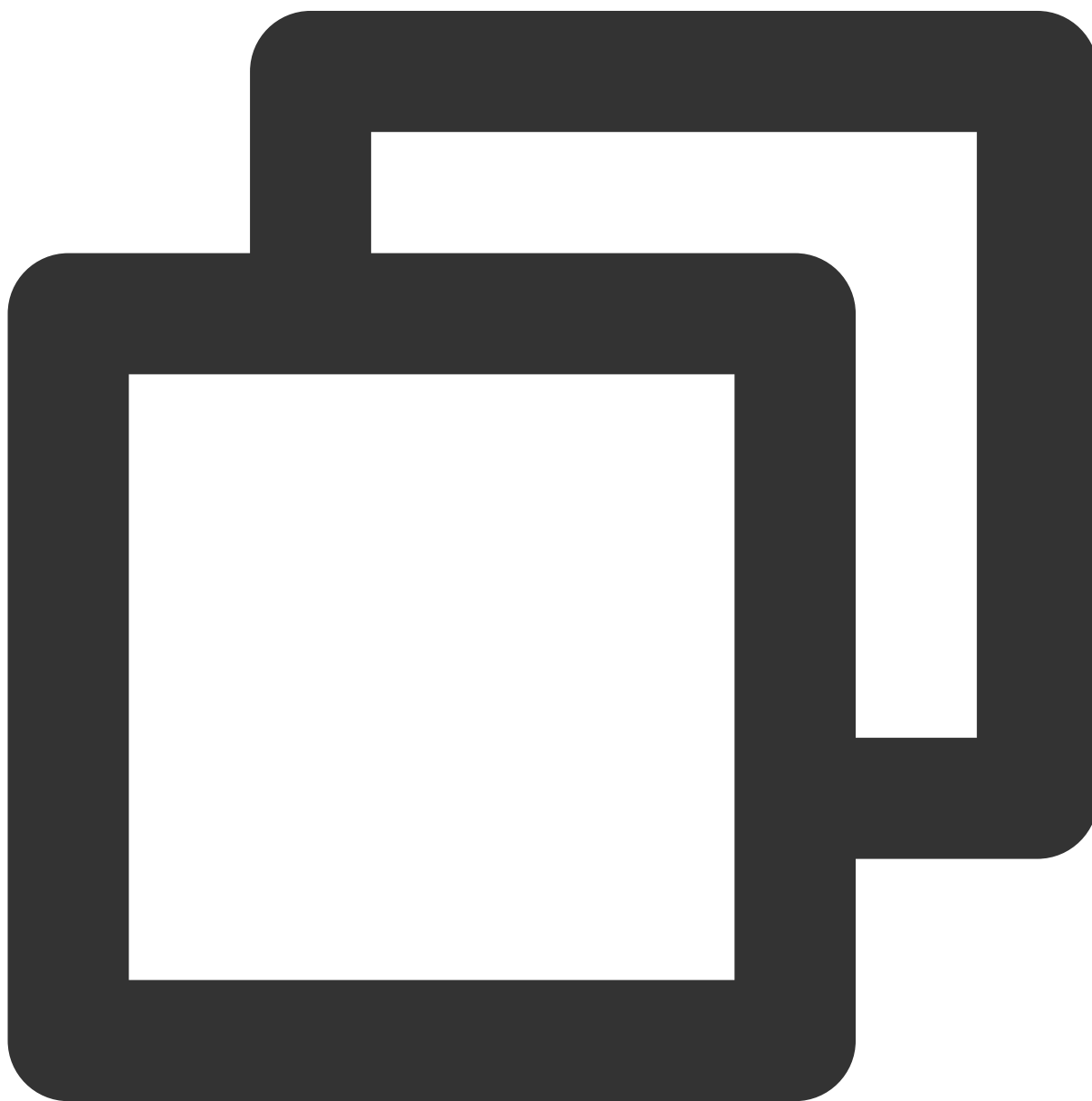
```
@ProxyService(proxy = MiniConfigProxy.class)
public class MiniConfigProxyImpl extends MiniConfigProxy {
    /**
     * 创建初始化配置信息
     * @return
     */
    @Override
    public MiniInitConfig buildConfig() {
        MiniInitConfig.Builder builder = new MiniInitConfig.Builder();
        MiniInitConfig config = builder
            .docLicenseKey("")//设置LicenseKey
```

```
        .build();  
    }  
}
```

媒体扩展 SDK

组件说明：提供 chooseMedia、previewMedia 的默认实现。

集成方法：按照如下的方式添加扩展库依赖：

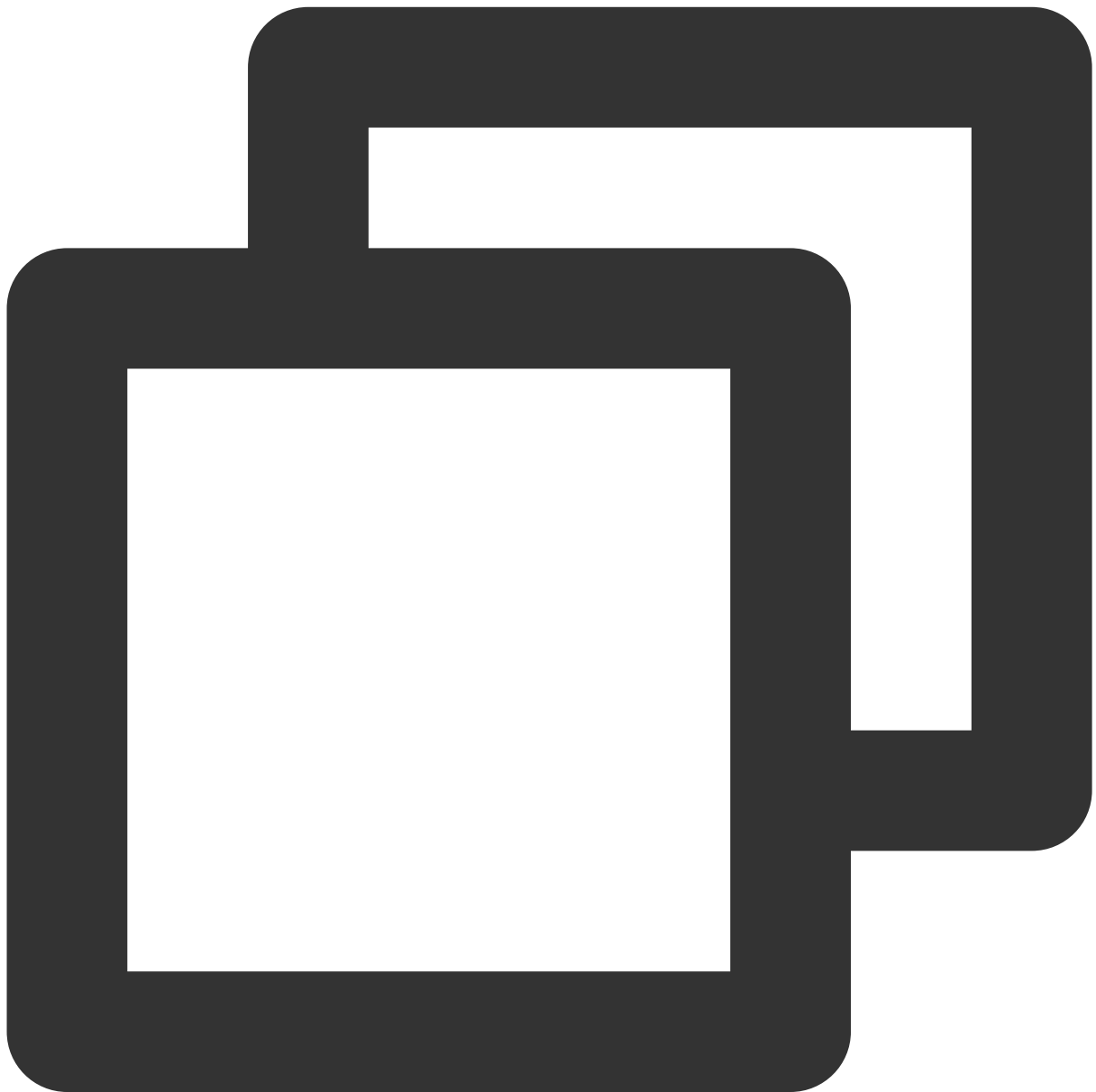


```
implementation 'com.tencent.tcmpp.android:mini_extra_media_support:${version}'//版本
```

实现 `MediaImageLoaderProxy` 代理，使用自定义的图片加载实现，用于 `mini_extra_media_support` 库的图片加载。

说明：

可以通过实现 `MediaChooseJsProxy` 代理，实现自定义的 `chooseMedia` 逻辑。



```
@ProxyService(proxy = MediaImageLoaderProxy.class)
public class CustomMediaImageLoaderProxy implements MediaImageLoaderProxy {
    private GlideImageEngine glideImageEngine = new GlideImageEngine();
```

```
@Override
public ImageEngine getCustomImageEngine() {
    return glideImageEngine;
}

static class GlideImageEngine implements ImageEngine {

    @Override
    public void loadPhoto(@NonNull Context context, @NonNull Uri uri, @NonNull
        Glide.with(context).load(uri).transition(withCrossFade()).into(imageView)

    }

    @Override
    public void loadGifAsBitmap(@NonNull Context context, @NonNull Uri gifUri,
        Glide.with(context).asBitmap().load(gifUri).into(imageView);

    }

    @Override
    public void loadGif(@NonNull Context context, @NonNull Uri gifUri, @NonNull
        Glide.with(context).asGif().load(gifUri).transition(withCrossFade()).in
```

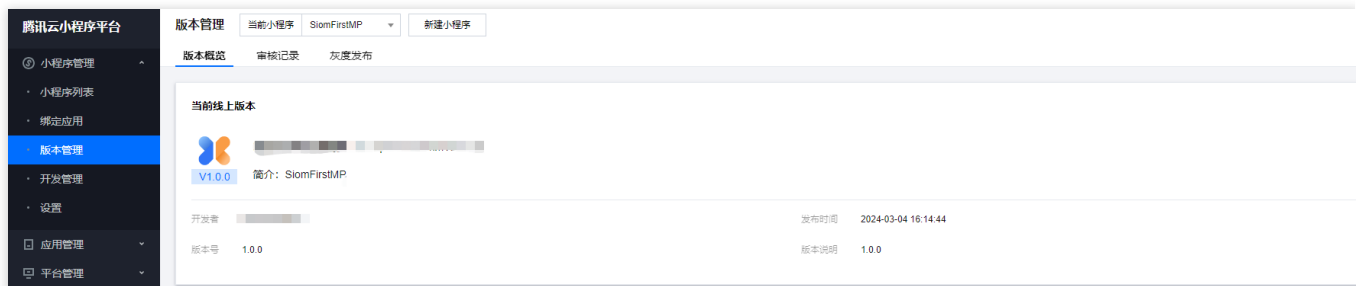

预置离线小程序

最近更新时间：2024-06-27 10:50:08

离线小程序是内置在宿主 App 内的小程序，需要将小程序包从控制台下载下来，导入到宿主 App 工程中，跟随 App 一起打包。在 App 使用过程中，用户打开某个内置的小程序，无需从后端拉取下载，即使在没有网络连接的情况下也能打开并运行的小程序。

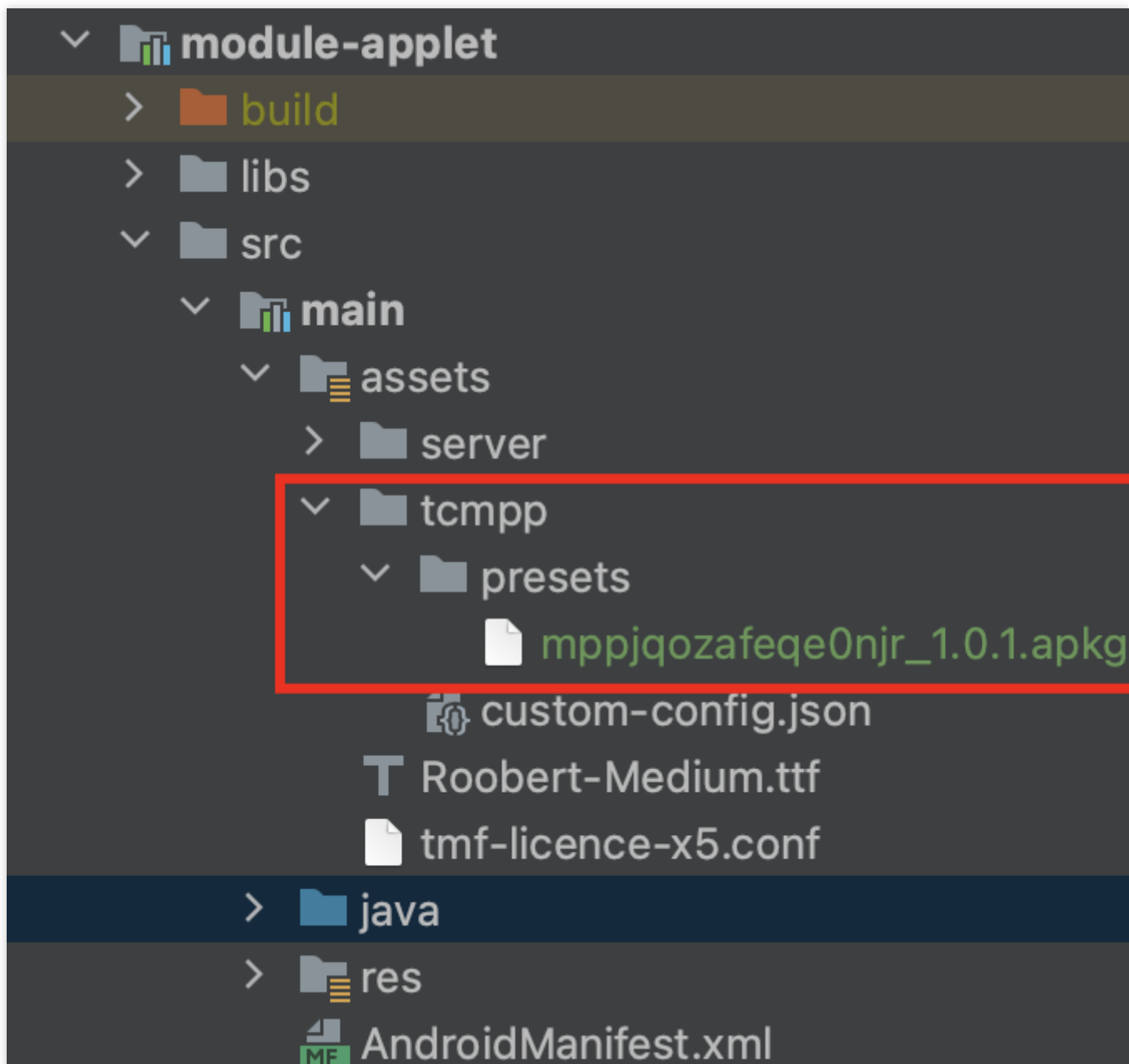
预置流程

1. 从控制台下载需要预置的小程序。



2. 将下载的小程序包拷贝到自定义的assets目录，离线小程序需严格遵守命名规则，不能随意修改。

离线小程序命名规则：`{miniAppId}_{miniAppVersion}.apkg`。



3. 在 SDK 初始化配置中指定上一步放置离线小程序的 asset 目录。

```
@ProxyService(proxy = MiniConfigProxy.class)
public class MiniConfigProxyImpl extends MiniConfigProxy {
    @Override
    public Application getApp() { return ModuleApplet.sApp; }

    1 usage  @Override
    public MiniInitConfig buildConfig() {
        //注意MiniInitConfig.Builder参数需要传入应用的Application
        MiniInitConfig.Builder builder = new MiniInitConfig.Builder
        return builder
            .assetPathOfPresets("tcmpp/presets")
            .build();
    }
}
```

注意事项

离线小程序也需要走正常的小程序绑定应用和发布流程，只有上线状态的小程序才可下载离线包；

离线小程序适用新版发布、版本回退、下架等版本管理逻辑，当线上版本与预置版本不一致时，客户端会正常拉取线上版本；

当小程序被下架后，预置在App中的对应离线小程序也无法正常使用。

Android 机制说明

最近更新时间：2024-06-27 10:49:49

Android 端 SDK 为什么要使用多进程来实现？

Android 端运行时 SDK 采用多进程机制实现，所谓多进程是指小程序与宿主 App 分隔开，各自运行在独立的进程中，进程之间互不干扰，通过跨进程通信相互传递数据。之所以选用多进程，原因主要有以下几点：

- 不占用宿主 App 的内存。系统会为小程序进程分独立的内存空间，小程序不会占用主进程的内存，因此 App 不用担心内存溢出等问题；
- 保证宿主 App 安全稳定运行。小程序进程中发生的一切异常都不会影响到宿主 App，即使小程序进程中发生了崩溃，宿主 App 照样能够继续正常运行；
- 更容易实现数据解耦。进程之间内存是天然隔离的，除非通过跨进程通信手段，否则一个进程中的数据永远都不会被另一个进程直接读写，因此不用担心数据耦合问题。

多进程如何实现？

因为 Android 没有提供动态创建进程的 API，所以采用为 Activity 绑定进程的方式来实现进程的创建，即在 AndroidManifest.xml 中为 Activity 指定 android:process 属性，当 Activity 被第一次启动时，系统会创建新的进程并将 Activity 所在的栈放到这个进程里面；

为了增强用户体验，需要将同时打开的不同小程序放在系统最近任务栏的不同位置展示，这样用户就能清楚地看到当前打开了哪些小程序并随意切换。为了实现这个效果，需要在过 AndroidManifest.xml 中为 Activity 设置 android:taskAffinity 属性，将不同小程序进程的 Activity 压入不同的任务栈。

小程序进程如何管理？

SDK 支持最多同时创建5个小程序进程，当小程序进程数量没有超过5个时，每启动一个新的小程序都会创建一个新的进程，当小程序个数达到最大数量之后，再启动新的小程序时，将会复用现有的小程序进程来打开小程序；

打开相同的小程序时，会先检查小程序是否已启动，如果已启动则会直接唤起对应的进程；

小程序进程的创建、复用、唤起等逻辑都是在主进程中完成的，由主进程统一调度。

小程序进程和主进程间如何通信？

小程序进程和宿主进程之间的双向通信是通过 AIDL 实现的，小程序进程和主进程绑定同一个远程服务，通过 Binder 机制获取到服务的代理对象，通过代理对象调用 AIDL 接口实现跨进程通信。

进程安全、线程安全问题如何处理？

目前小程序运行过程中需要写入的资源均存放在应用沙盒内小程序对应的目录下，没有对全局共享资源的写操作，因此不存在进程间竞争同一系统资源的问题。对于将来可能存在的进程安全问题，在开发过程中会使用进程安全的方式来读写数据，例如使用文件锁访问文件、通过 ContentProvider 代替 SharedPreferences 共享数据等；

进程间并不存在直接的线程安全问题，只有不同的小程序进程通过跨进程通信并发访问主进程的内存时，或者进程内部并发读写内存时才会带来线程安全问题，SDK 通过锁、同步方法、java 并发类集等方式来解决多线程安全相关

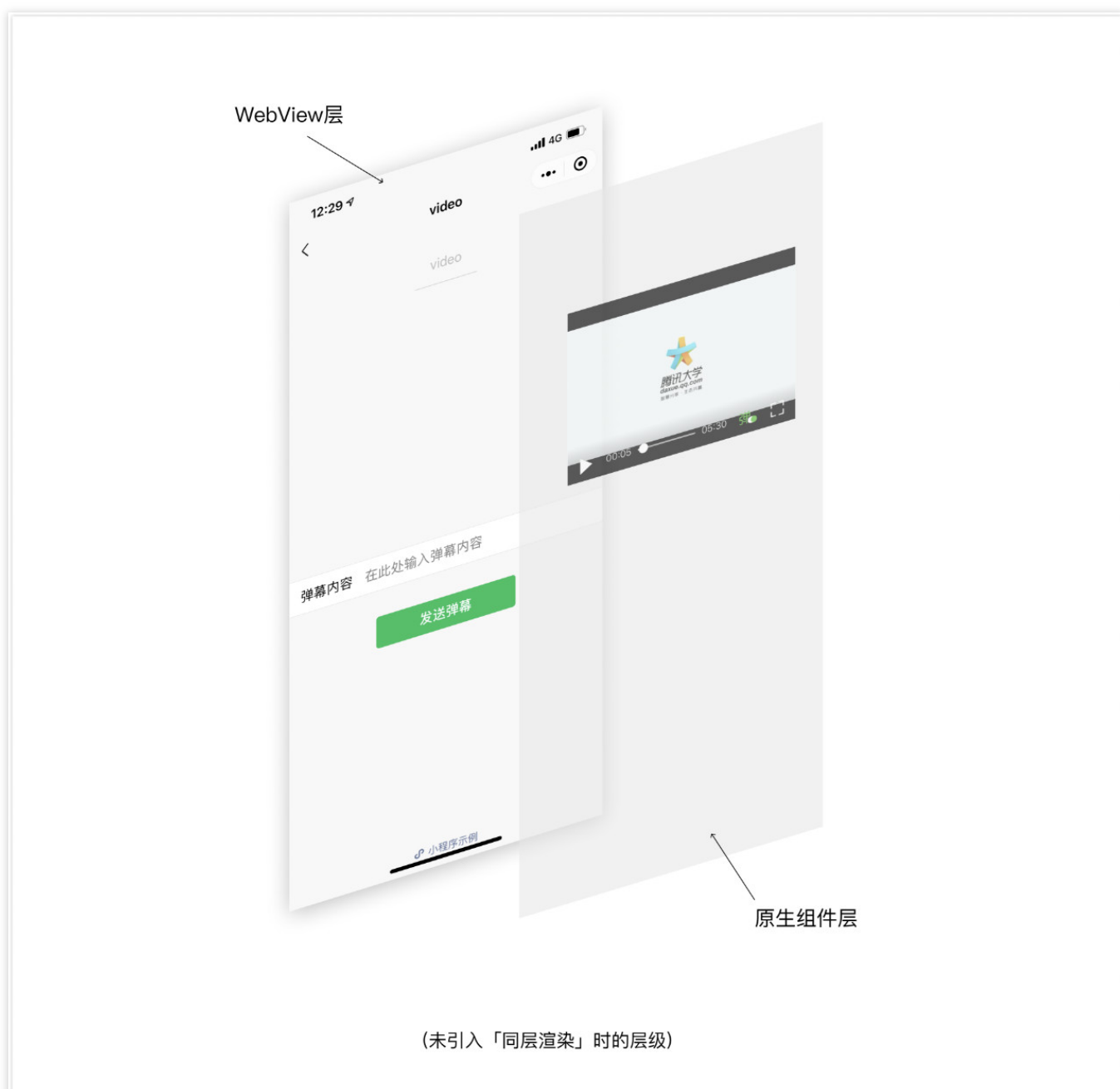
问题。

Android 常见问题

最近更新时间：2024-06-27 16:41:36

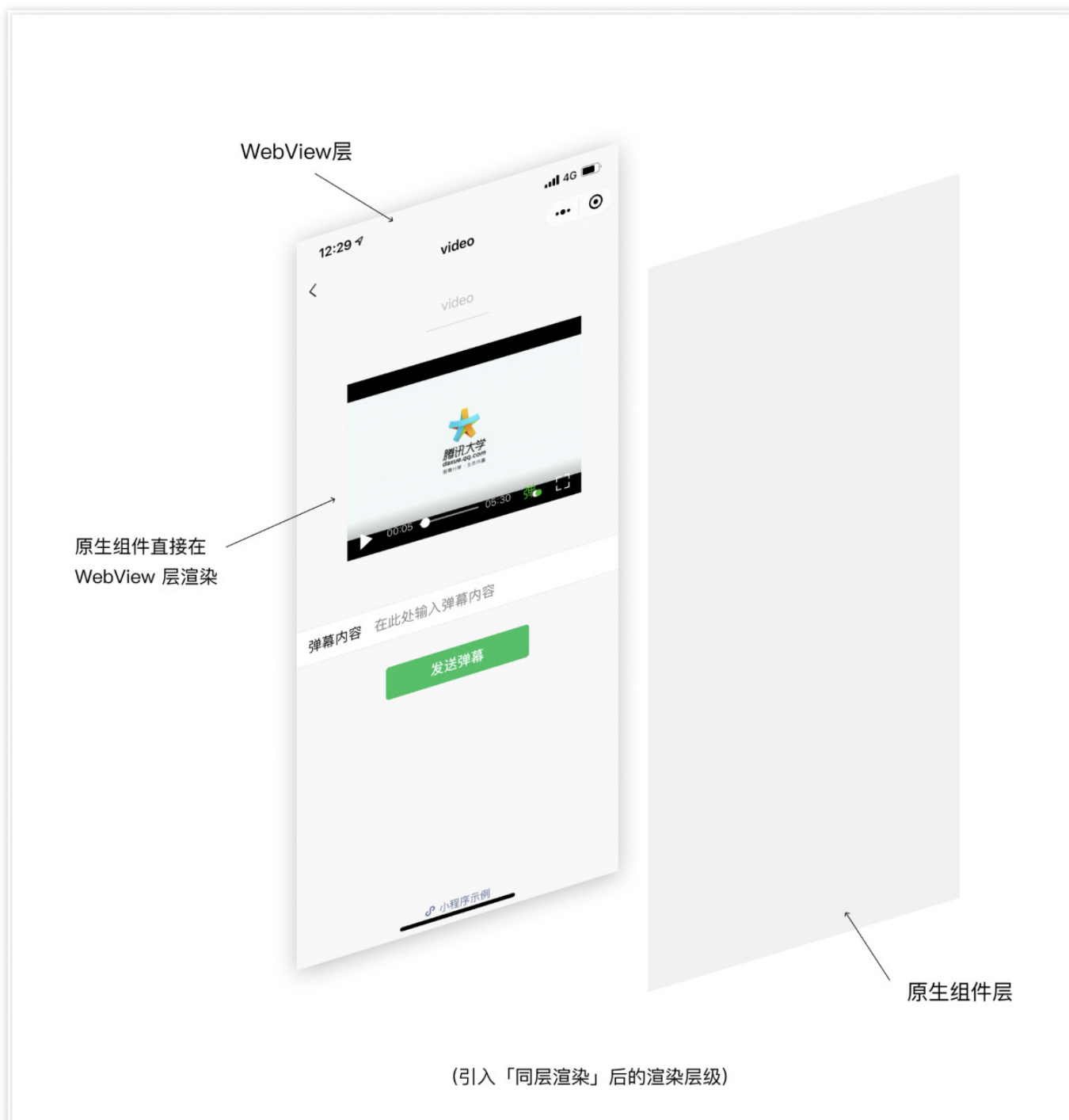
什么是同层渲染？

小程序的内容大多是渲染在 **WebView** 上的，如果把 **WebView** 看成单独的一层，那么由系统自带的这些原生组件则位于另一个更高的层级。两个层级是完全独立的，因此无法简单地通过使用 **z-index** 控制原生组件和非原生组件之间的相对层级。正如下图所示，非原生组件位于 **WebView** 层，而原生组件及 **cover-view** 与 **cover-image** 则位于另一个较高的层级：



那么「同层渲染」顾名思义则是指通过一定的技术手段把原生组件直接渲染到 **WebView** 层级上，此时「原生组件层」已经不存在，原生组件此时已被直接挂载到 **WebView** 节点上。你几乎可以像使用非原生组件一样去使用「同层

渲染」的原生组件，比如使用 `view`、`image` 覆盖原生组件、使用 `z-index` 指定原生组件的层级、把原生组件放置在 `scroll-view`、`swiper`、`movable-view` 等容器内，通过 `WXSS` 设置原生组件的样式等等。启用「同层渲染」之后的界面层级如下图所示：



同层渲染的实现原理？

小程序在 Android 端采用 chromium 作为 WebView 渲染层，Android 端的 WebView 是单独进行渲染，经渲染后的 WebView 是一个完整的视图，因此需要采用其他的方案来实现「同层渲染」。经过我们的调研发现，chromium 支持 WebPlugin 机制，WebPlugin 是浏览器内核的一个插件机制，主要用来解析和描述 `embed` 标签。Android 端的同层渲染就是基于 `embed` 标签结合 chromium (x5内核定制) 内核扩展来实现的。

```
<embed id="web-plugin" type="plugin/video" width=300 height=300>
```

Android 端「同层渲染」的大致流程如下：

WebView 侧创建一个 embed DOM 节点并指定组件类型；

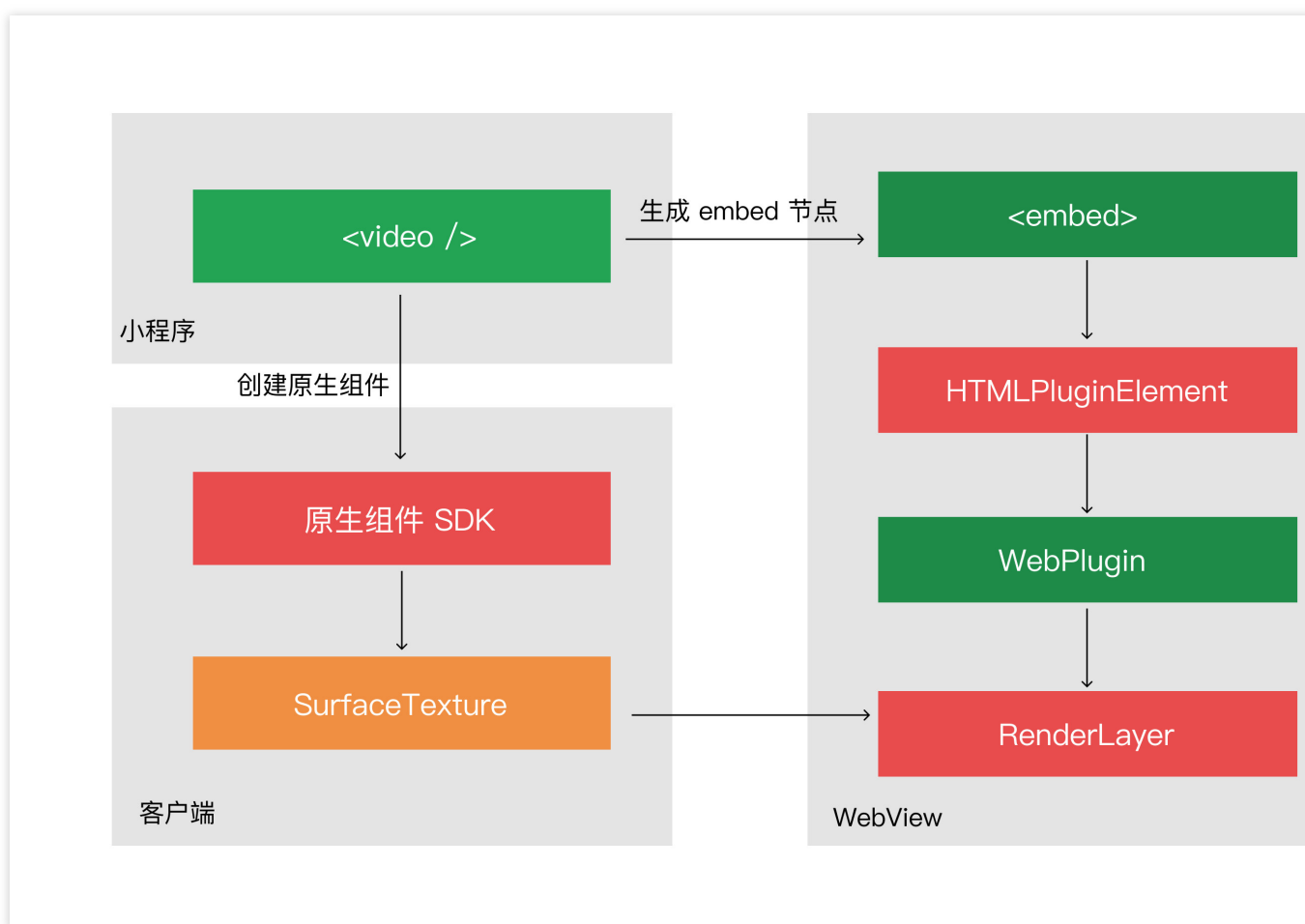
chromium 内核会创建一个 WebPlugin 实例，并生成一个 RenderLayer；

Android 客户端初始化一个对应的原生组件；

Android 客户端将原生组件的画面绘制到步骤2创建的 RenderLayer 所绑定的 SurfaceTexture 上；

通知 chromium 内核渲染该 RenderLayer；

chromium 渲染该 embed 节点并上屏。



通过这种方式可以用于 map、video、canvas、camera、textarea 和 input 等原生组件的渲染。

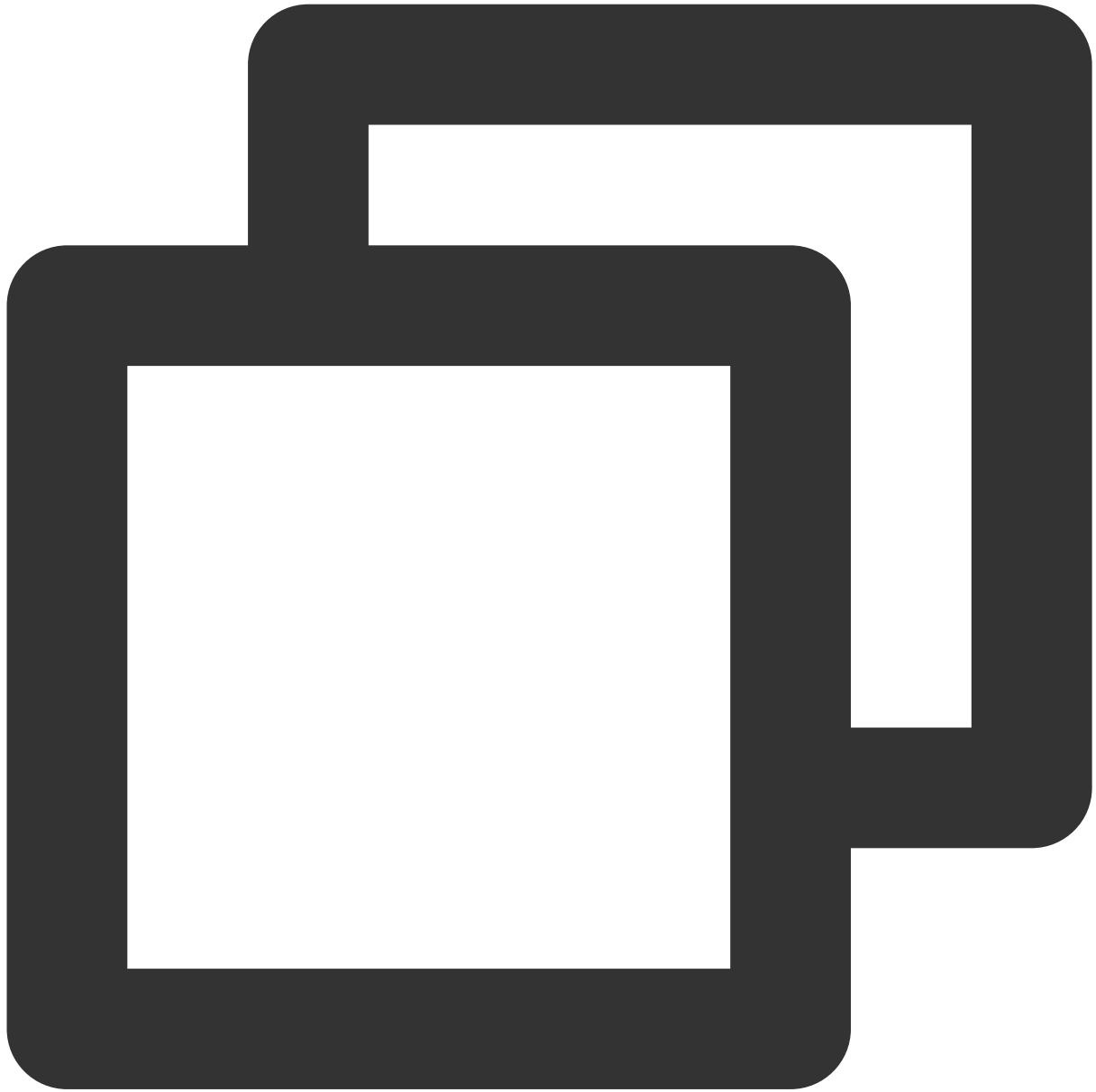
小程序启动失败排查

小程序启动失败，可能有如下几种原因：

原因一：配置文件路径错误，`configAssetName` 设置的是 `assets` 目录下文件的完整路径，如果配置文件在子目录下需要追加目录路径，例如：`server/tcmpp-android-configurations.json`。

原因二：不允许修改小程序配置文件内容，否则小程序无法正常运行。

原因三：配置文件中的 **packageName** 必须与应用的 **applicationId** 保持一致，否则 App 运行失败，因为初始化时会校验配置文件中的 **packageName**，可以通过如下设置不进行包名校验。

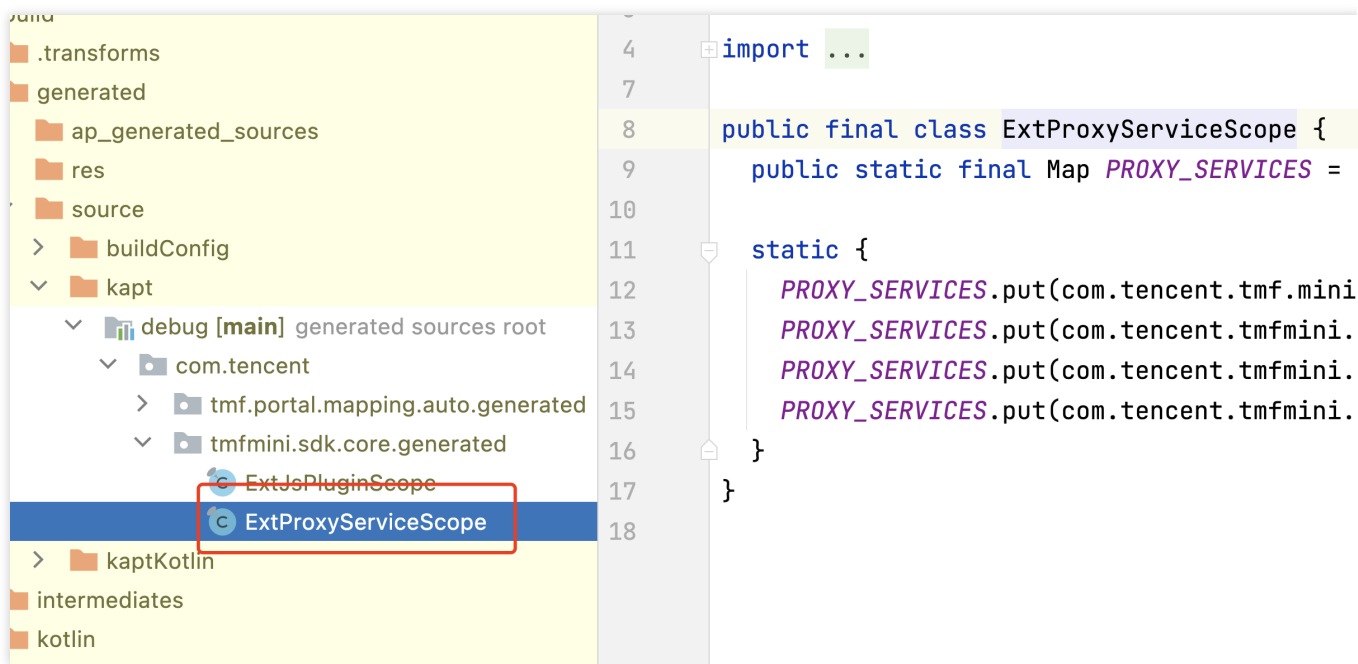


```
MiniInitConfig config = builder
    .verifyPkg(false) //忽略包名校验
    .build();
```

```
{
  "channel": "Android",
  "customId": "9ab50ac0-be06-11ec-a34e-278d66d394b5",
  "material": "01X7UBa0sAdvWstZ0sTT0mdoDnZghwoG145kRt8B3i0vvTj31DVuIuXrmEZfthnyryyJngGkBm1I4AckFtf",
  "packageName": "com.tencent.tmf.miniapp.demo",
  "productId": "7686",
  "qapm": {
    "appKey": "None",
    "url": "https://tmfqapm.qq.com:30026"
  },
  "shark": {
    "appKey": "app-dfrh27n0vw",
    "asymEnc": 1,
    "httpUrl": "https://tmfmp.qq.com:30013/",
    "symEnc": 2,
    "tcpHost": "42.194.253.71",
    "tcpPort": 30014
  }
}
```

```
android {
    compileSdkVersion 30
    defaultConfig {
        applicationId "com.tencent.tmf.miniapp.de
        //feed back 测试
        //      applicationId "tmf.tencent.tmf.demo.db"
        minSdkVersion 21
        targetSdkVersion 28
        versionCode 1
        versionName "1.0"
        ndk { NdkOptions it -> abiFilters "arm64-v8a"
```

原因四：确定上面初始化注解 @ProxyService 是否生成了如下类 ExtProxyServiceScope：

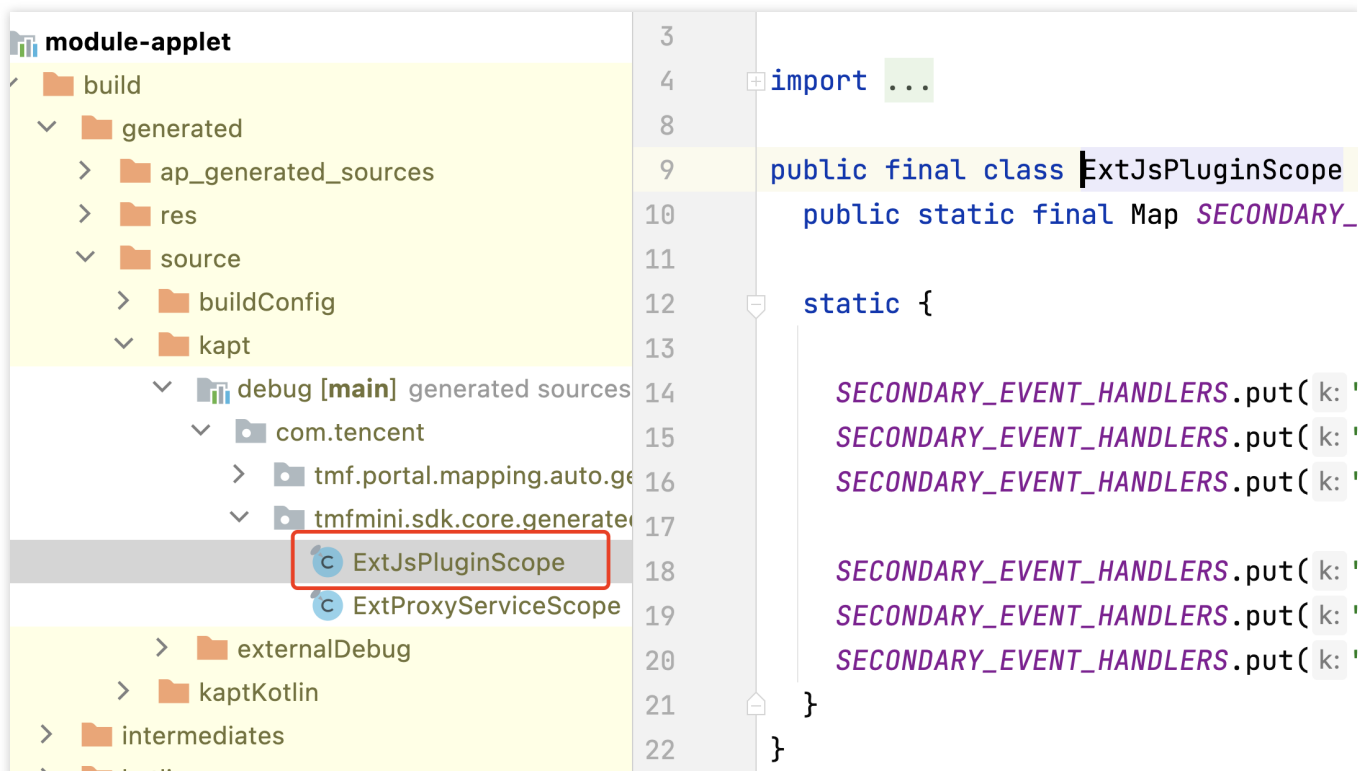


SDK 如何确保隐私合规？

小程序 SDK 的初始化是在开发者调用 TmfMiniSDK 类的方法时调用的，所以在用户同意隐私合规授权之后，再调用 TmfMiniSDK 类中提供的方法即可。

自定义小程序 API 出现错误，怎么排查？

在编译路径中，检查是否生成 XxxJsPluginScope，如下图：



检查客户端定义的事件与小程序调用方法名是否一致，自定义小程序API的事件名称区分大小写。

小程序域名和隐私 API 校验逻辑

小程序使用过程中会校验接口请求域名的合法性，以及如果在管理后台设置了隐私 API 也会进行授权校验，但在如下场景下不进行校验：

运行小程序是非正式版本且小程序开启了[小程序调试](#)。

模块化工程支持

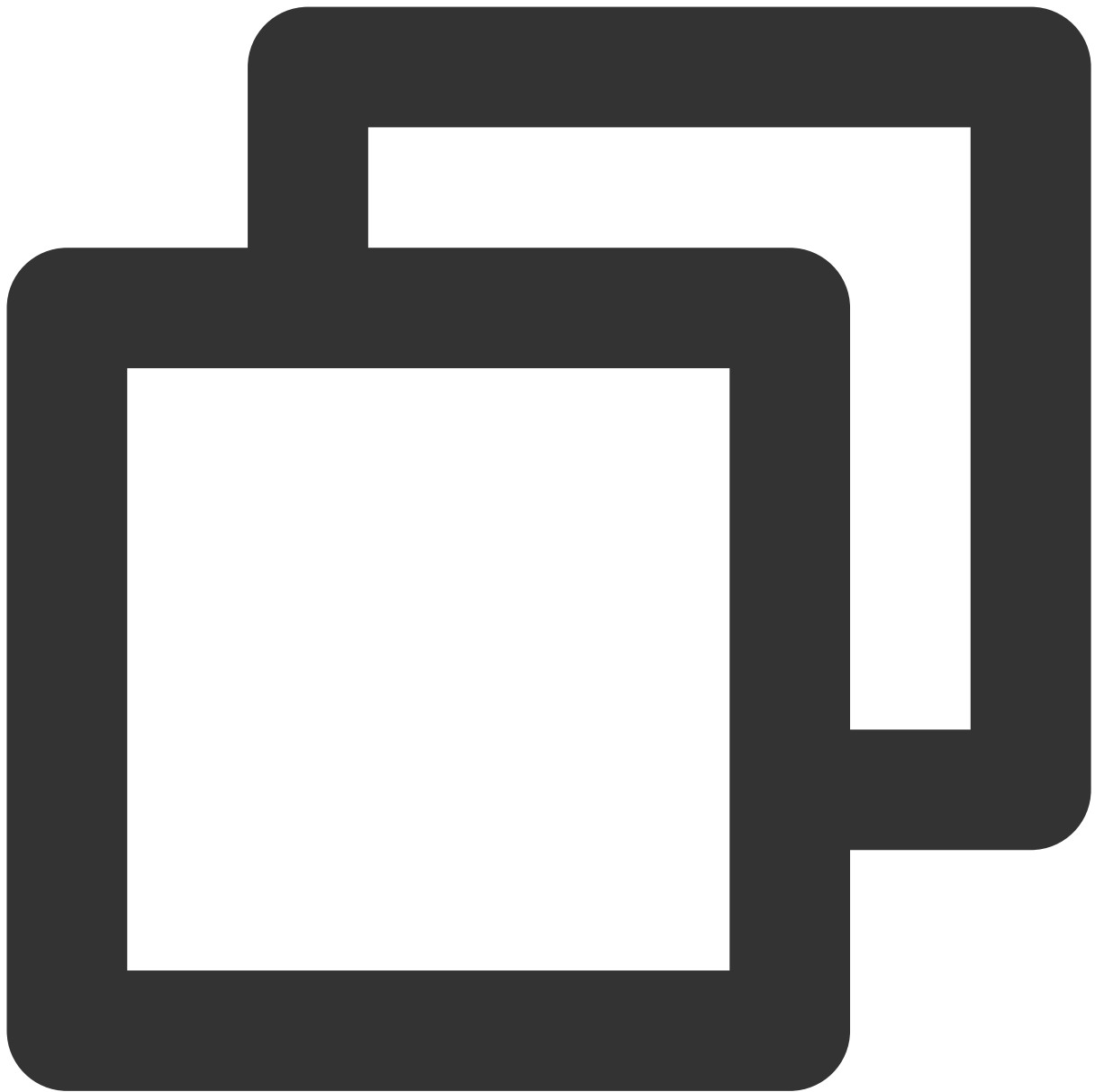
当开发者在模块化工程中多个模块同时使用了注解 `@JsPlugin` 或 `@ProxyService` 时，Make Project 工程时会出现如下错误：

```
Execution failed for task ':DEMO:mergeLibDexDebug'.
> A failure occurred while executing com.android.build.gradle.internal.tasks.Workers$Action
> com.android.builder.dexing.DexArchiveMergerException: Error while merging dex archiv
Learn how to resolve the issue at https://developer.android.com/studio/build/dependencies#duplicate\_classes.
Program type already present: com.tencent.tmfmini.sdk.core.generated.ExtJsPluginScop

* Try:
Run with --stacktrace option to get the stack trace. Run with --info or --debug option to
output. Run with --scan to get full insights.
```

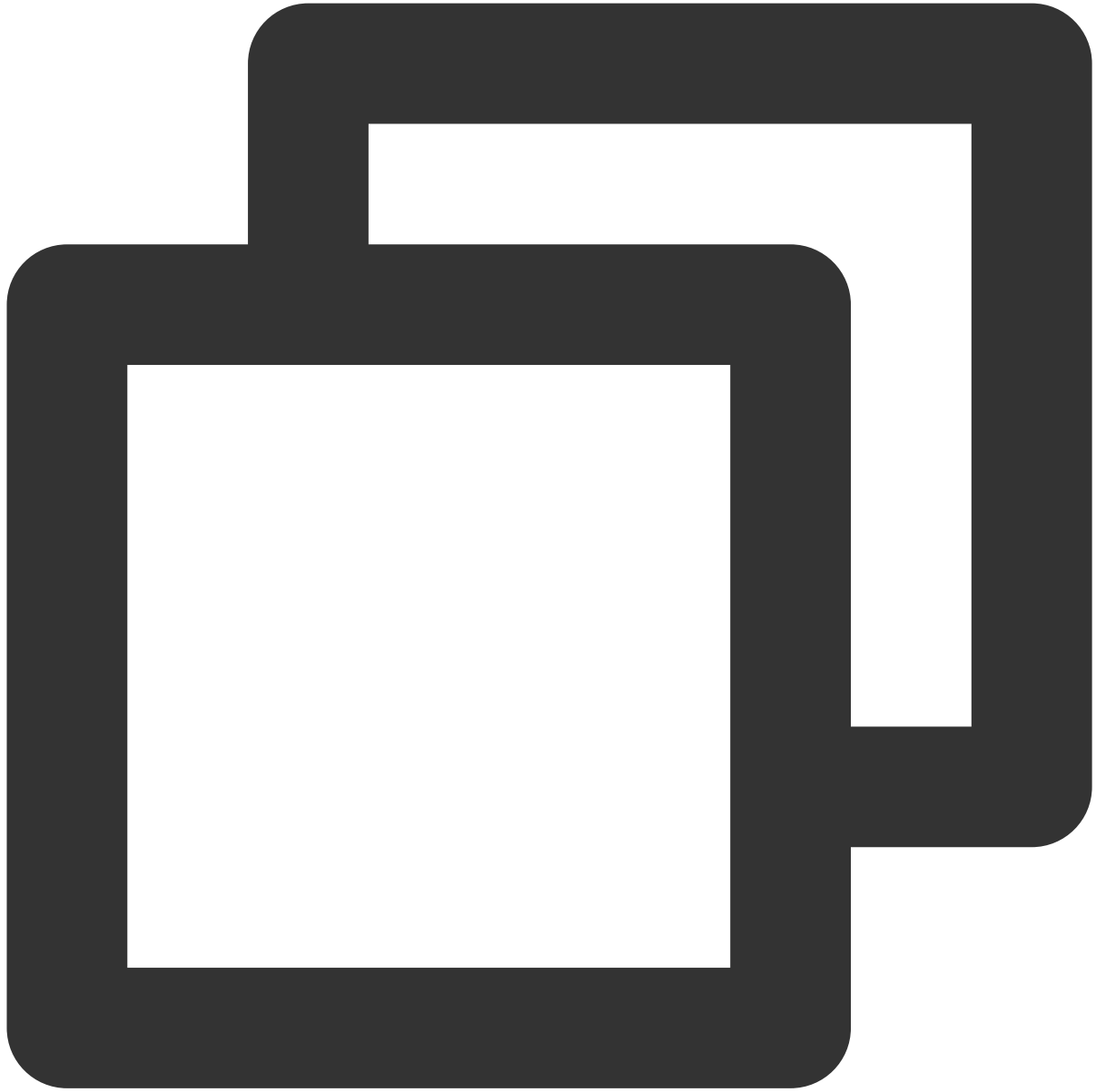
需要按照如下配置支持多模块化工程：

1. 在每个使用了 `@JsPlugin` 或 `@ProxyService` 注解的 module 的 build.gradle 中添加如下代码：



```
android {  
    defaultConfig {  
        javaCompileOptions {  
            annotationProcessorOptions {  
                //配置模块名：开发者自己定义唯一名称, 模块名遵循android类名定义规范  
                arguments = [tcmppModuleName: "Demo"]  
            }  
        }  
    }  
}
```

2. 初始化代码中注册模块：

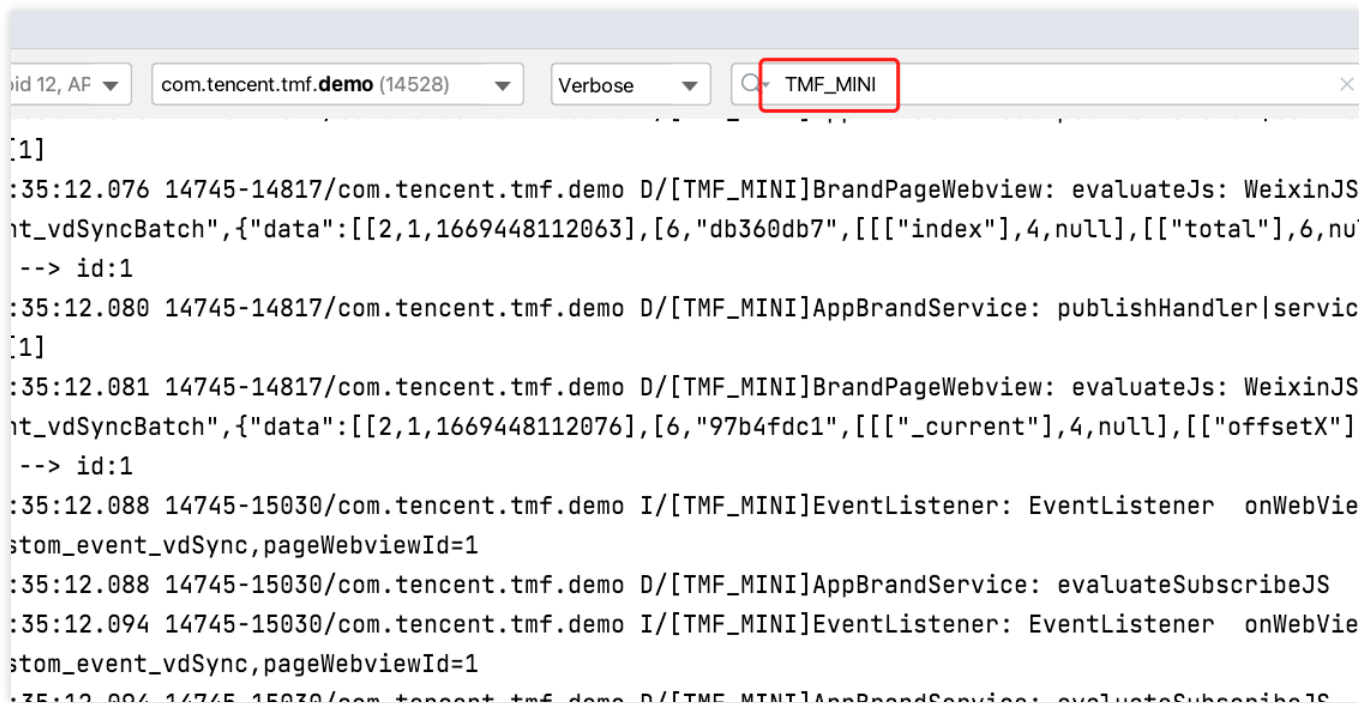


```
@ProxyService(proxy = MiniConfigProxy.class)
public class MiniConfigProxyImpl extends MiniConfigProxy {
    @Override
    public MiniInitConfig buildConfig() {
        MiniInitConfig.Builder builder = new MiniInitConfig.Builder();

        //将上面定义的所有module都进行注册，registerModule参数值与上面tcmppModuleName定义的
        return builder
    }
}
```

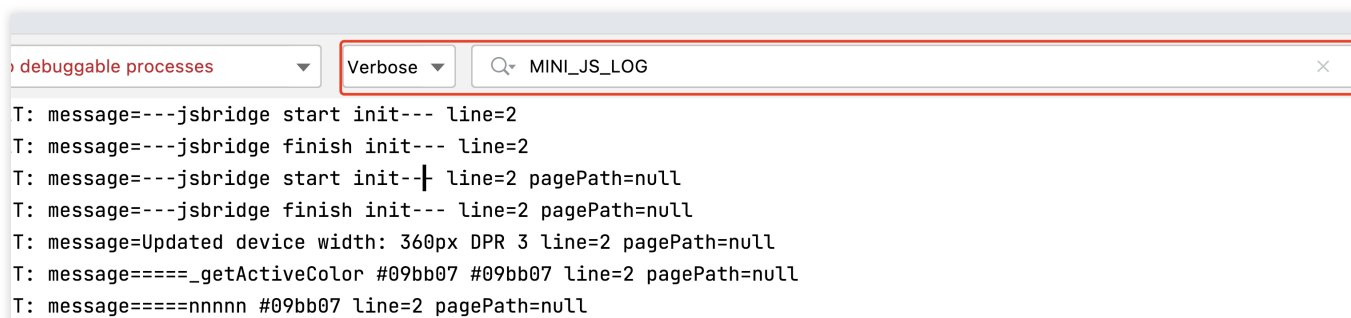
如何查看 SDK 日志输出？

1. 开发者可以在开发工具上，通过关键字 TMF_MINI 过滤得到 SDK 日志。

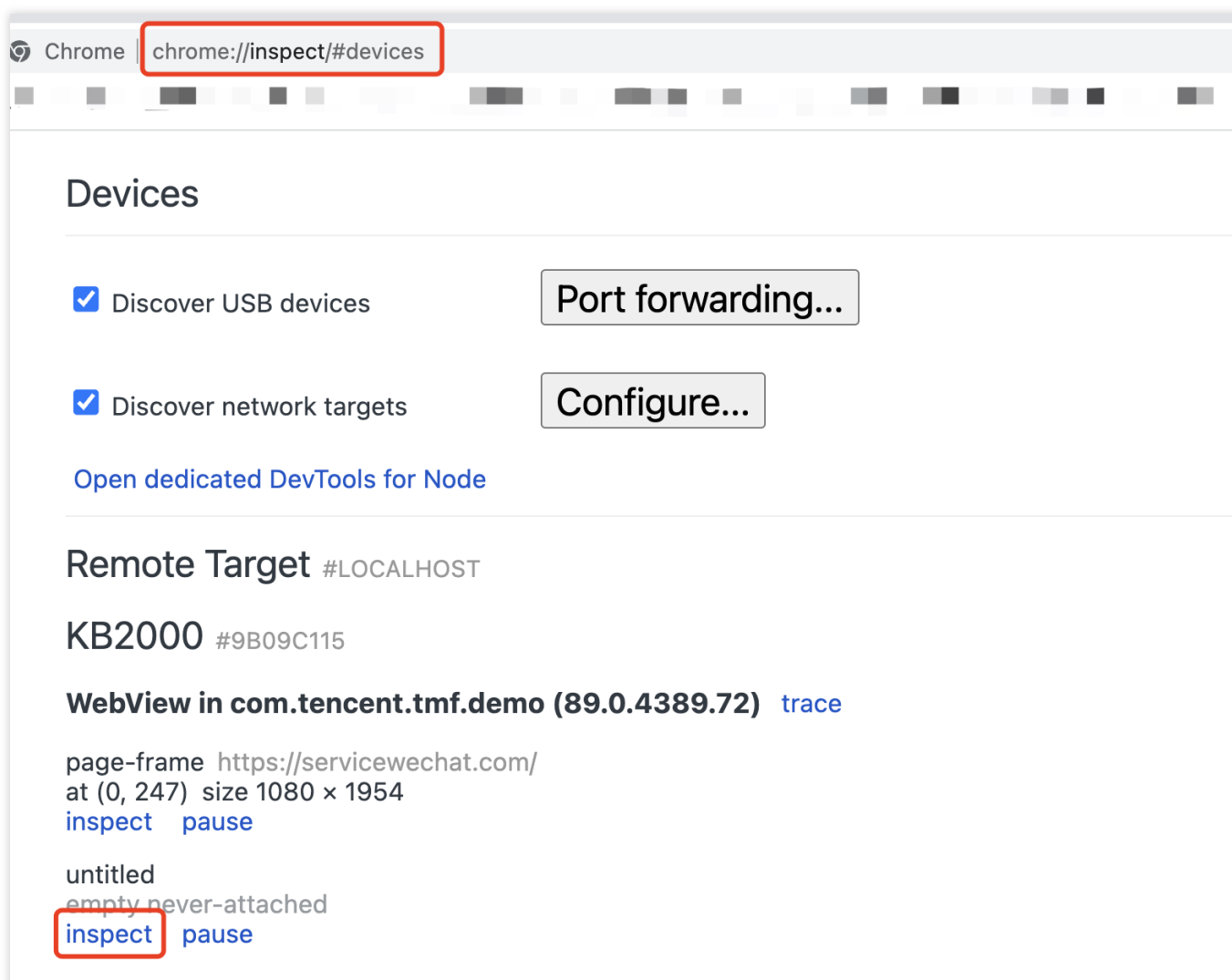


2. 查看小程序 JS 错误日志。

方式一：开发者可以通过关键字 MINI_JS_LOG 过滤得到JS日志。

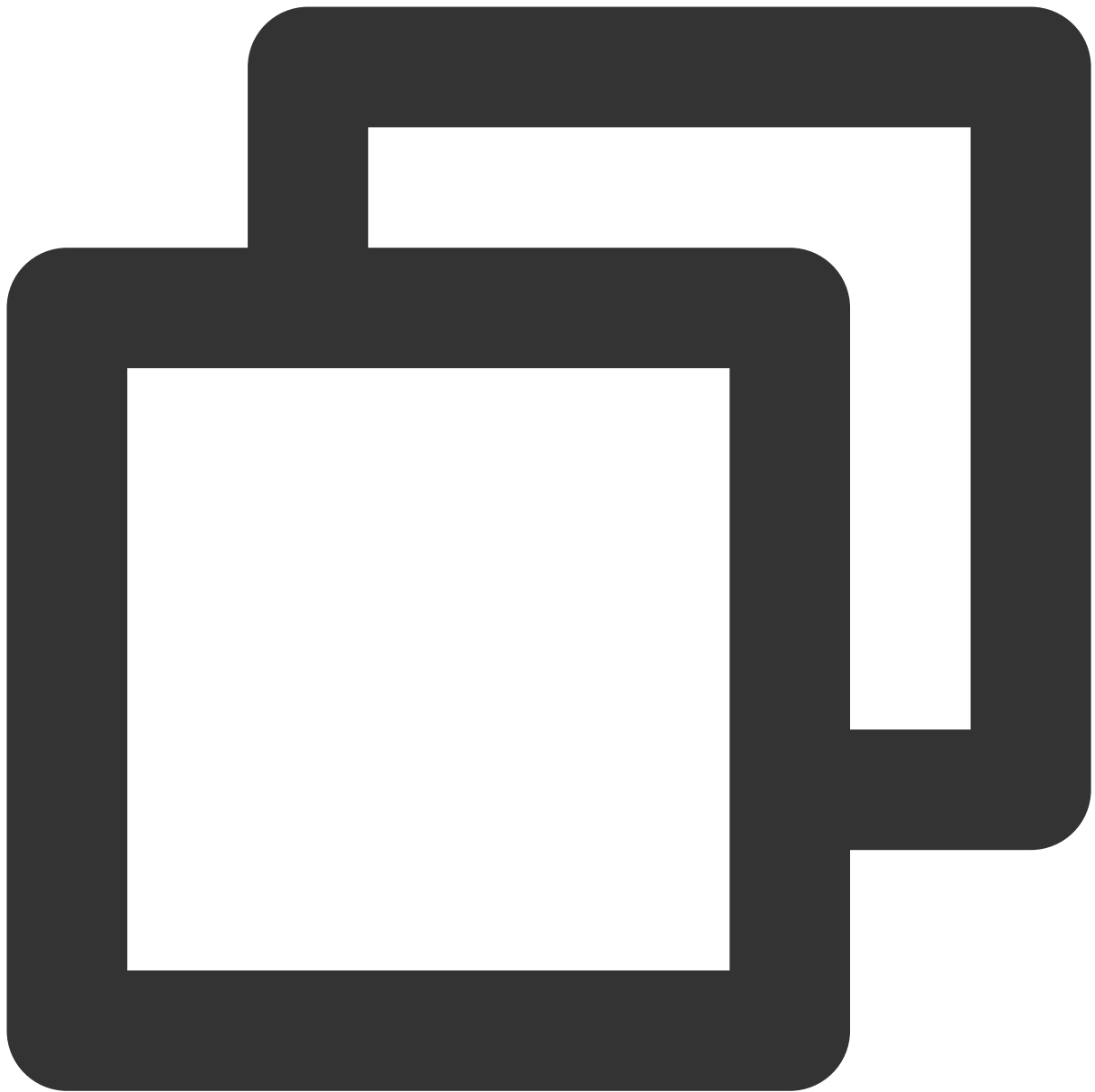


方式二：chrome 调试小程序的 JS 是否有错误。



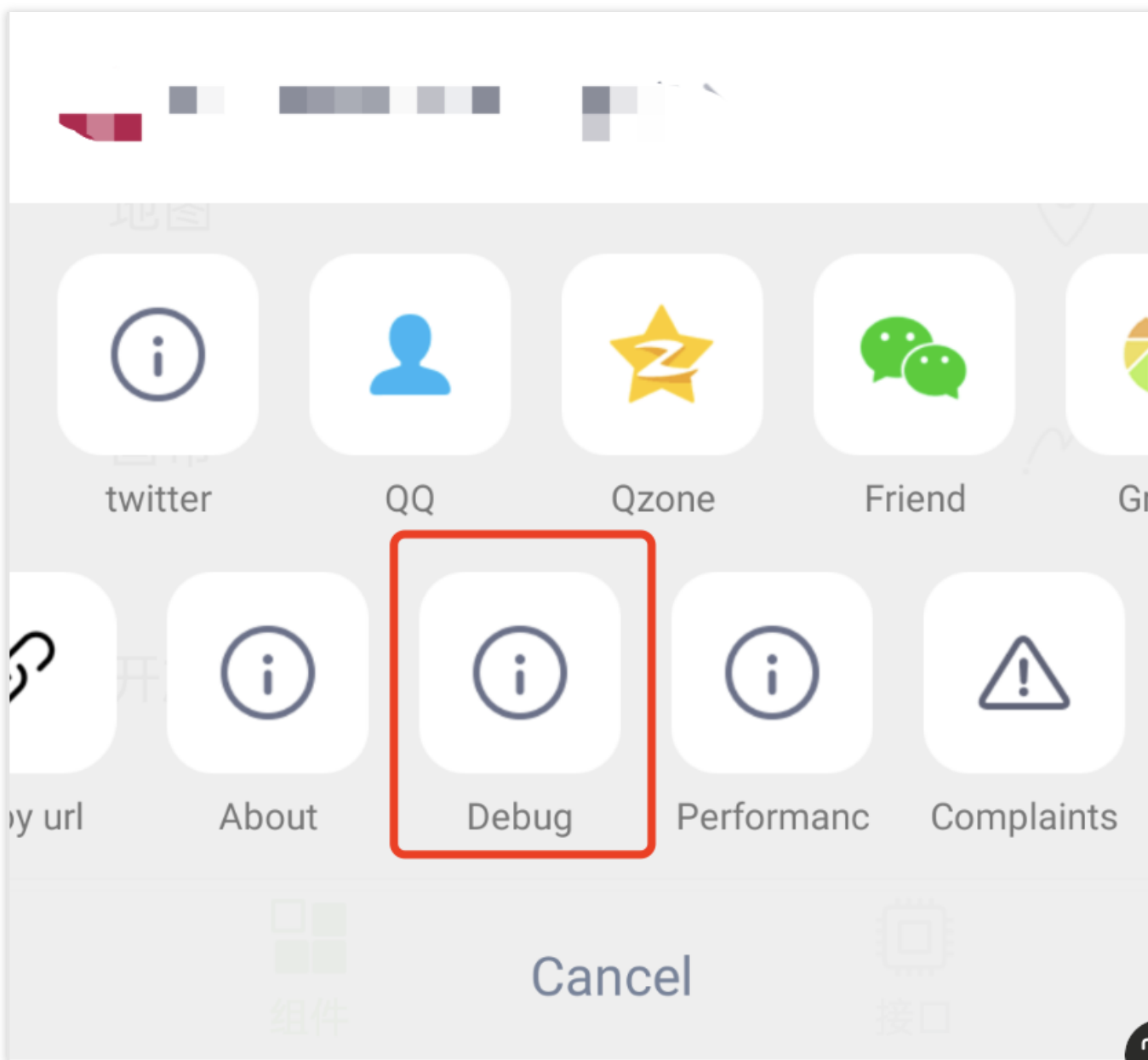
如何打开小程序的调试模式？

为了方便小程序调试查看 log 日志，客户端需要开启小程序调试入口，参考[小程序宿主自定义](#)-"自定义胶囊"getMoreItems 实现。



```
/**
 * 返回胶囊更多面板的按钮，扩展按钮的ID需要设置为[100, 200]这个区间中的值，否则，添加无效
 * 注：此方法在小程序进程调用
 * @param builder
 * @return
 */
public ArrayList<MoreItem> getMoreItems (MoreItemList.Builder builder) {
    builder.addDebug ("调试", icon) //设置控制面板调试按钮
    return builder.build();
}
```

设置后小程序控制面板会出现调试按钮，单击开启调试模式后，重启小程序进入调试模式。



小程序 SDK 会主动清理小程序包缓存吗？

不会，需要开发者自行调用小程序 SDK 提供的小程序删除方法进行清理（[删除小程序](#)）；如果未主动清理，宿主应用的缓存会随着使用小程序数量的增大而增大。

Android 错误码

最近更新时间：2024-07-29 16:44:01

错误码	含义
0	成功
-11001	shark网络错误
-11002	服务端返回code错误
-11003	服务端返回response为空
-11004	服务端返回小程序更新类型有问题
-11005	服务端返回数据解析异常
-11006	小程序不存在或已下架
-11007	达到月活上限
-11008	资源限额
-11011	特定接口请求频率过高
-11012	接口请求频率过高
-12001	shark实例空
-12002	预览小程序需要先登录
-12003	数据解析异常
-12004	扫码异常
-12005	小程序信息缺失
-12006	扫码错误
-12007	非tmf小程序二维码
-12008	小程序appId为空
-12009	businessId为空
-12010	小程序启动异常

-12011	json解析异常
-12012	小程序下载失败
-12013	小程序解析失败

Android 隐私合规

最近更新时间：2024-06-27 10:49:08

小程序 SDK 涉及的系统权限信息：

SDK名称	涉及的系统权限	权限使用描述
mini_core（含网关）	android.permission.INTERNET android.permission.READ_EXTERNAL_STORAGE android.permission.WRITE_EXTERNAL_STORAGE android.permission.READ_PHONE_STATE android.permission.ACCESS_WIFI_STATE android.permission.ACCESS_NETWORK_STATE android.permission.WAKE_LOCK android.permission.READ_MEDIA_AUDIO android.permission.CHANGE_NETWORK_STATE	访问网络以完成小程序容器和后台接口交互。 访问本地存储以实现小程序数据的持久化
mini_extra_qrcode	android.permission.CAMERA android.permission.WRITE_EXTERNAL_STORAGE android.permission.FLASHLIGHT	需要访问摄像头以实现扫码功能， 访问本地图片以实现二维码图片识别
mini_extra_v8	无	
mini_extra_dynamic_x5	无	
mini_extra_static_x5	无	
mini_extra_static_x5_new	无	
mini_extra_public_x5	无	
mini_extra_map	无	
mini_extra_google_map	android.permission.ACCESS_FINE_LOCATION	访问准确位置信息以提供定位能力
mini_extra_huawei_map	android.permission.INTERNET android.permission.ACCESS_NETWORK_STATE android.permission.CHANGE_WIFI_STATE android.permission.ACCESS_COARSE_LOCATION android.permission.ACCESS_FINE_LOCATION	华为地图服务要求集成
mini_extra_trtc_live	无	
mini_extra_nfc	android.permission.NFC	拥有该权限，才能使用

	android.permission.NFC_TRANSACTION_EVENT	设备提供的NFC能力
mini_extra_lbs	android.permission.ACCESS_FINE_LOCATION	需要访问设备位置信息，以提供定位相关的API能力
mini_extra_bluetooth	android.permission.BLUETOOTH_ADMIN android.permission.BLUETOOTH_SCAN android.permission.BLUETOOTH_ADVERTISE android.permission.BLUETOOTH_CONNECT	需要提供蓝牙权限，以扩展蓝牙相关API的能力
mini_extra_contact	android.permission.WRITE_CONTACTS android.permission.READ_CONTACTS	通讯录读取和写入权限，以扩展小程序通讯录访问相关API
mini_extra_soter	android.permission.USE_FINGERPRINT android.permission.USE_BIOMETRIC	生物识别相关权限，以扩展小程序的生物识别相关API
mini_extra_clipboard	无	
mini_extra_calendar	android.permission.READ_CALENDAR android.permission.WRITE_CALENDAR	日历访问权限，以提供小程序日程管理相关的API
mini_extra_lamemp3	无	
mini_extra_doc	无	
mini_extra_wifi	android.permission.ACCESS_FINE_LOCATION android.permission.ACCESS_WIFI_STATE android.permission.CHANGE_WIFI_STATE android.permission.ACCESS_NETWORK_STATE android.permission.CHANGE_NETWORK_STATE	需要网络状态和修改网络状态信息权限，以实现小程序Wi-Fi状态获取、Wi-Fi管理相关接口的能力
mini_extra_network	android.permission.INTERNET android.permission.ACCESS_NETWORK_STATE	需要网络及网络状态权限，以提供小程序网络状态相关的API扩展
mini_extra_media_support	无	
tbsscore	android.permission.INTERNET	
dynamicx5	android.permission.INTERNET	
com.tencent.tbs.tbssdk	由三方提供，未知。	
tbs-doc-support	由三方提供，未知。	

更多信息，可参见 [隐私合规说明](#)。

iOS

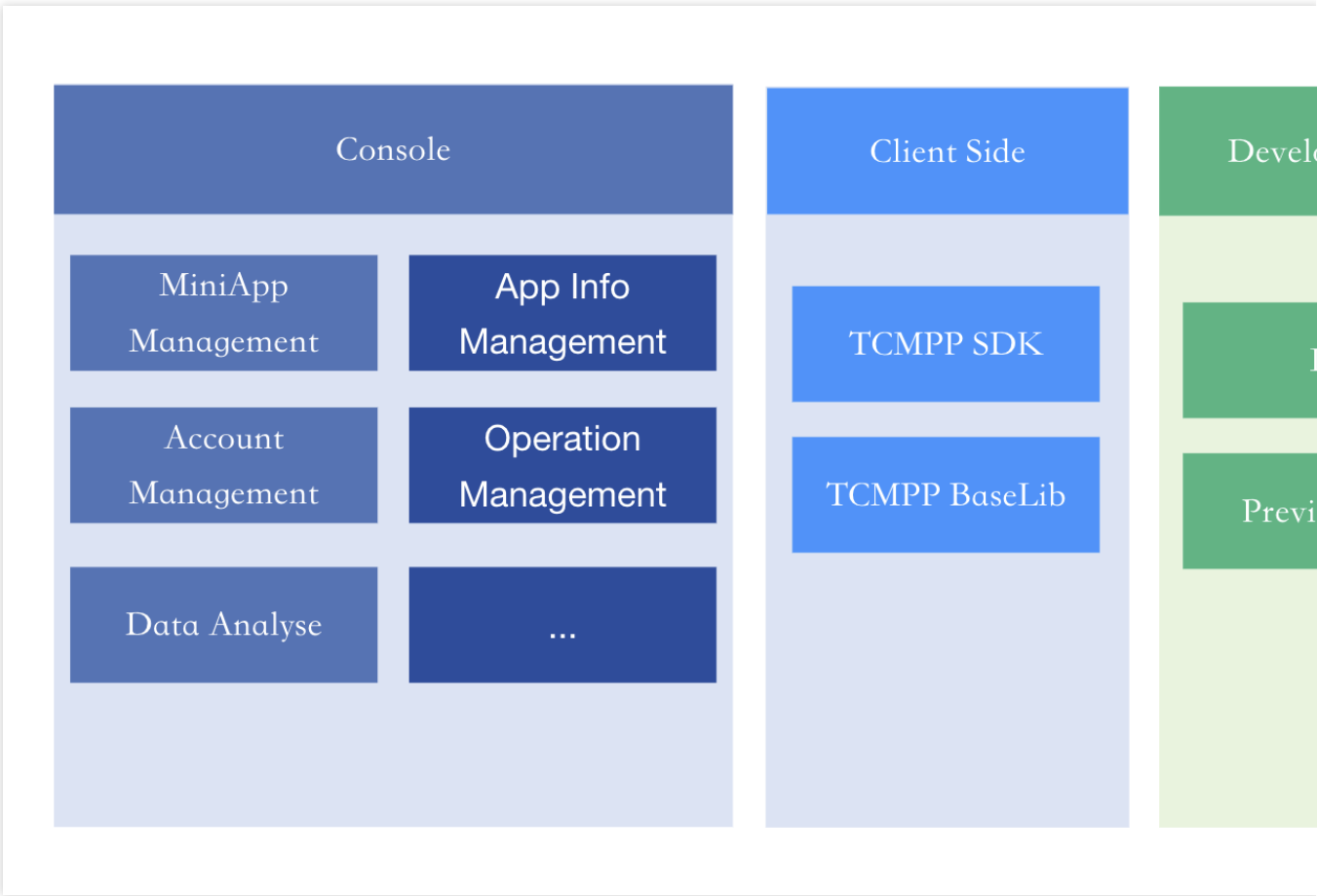
iOS SDK 说明

SDK 介绍

最近更新时间：2024-07-03 18:00:11

TCMPP 介绍

TCMPP 由管理控制台，客户端 SDK 和小程序开发工具（TCMPP IDE）三个部分组成。

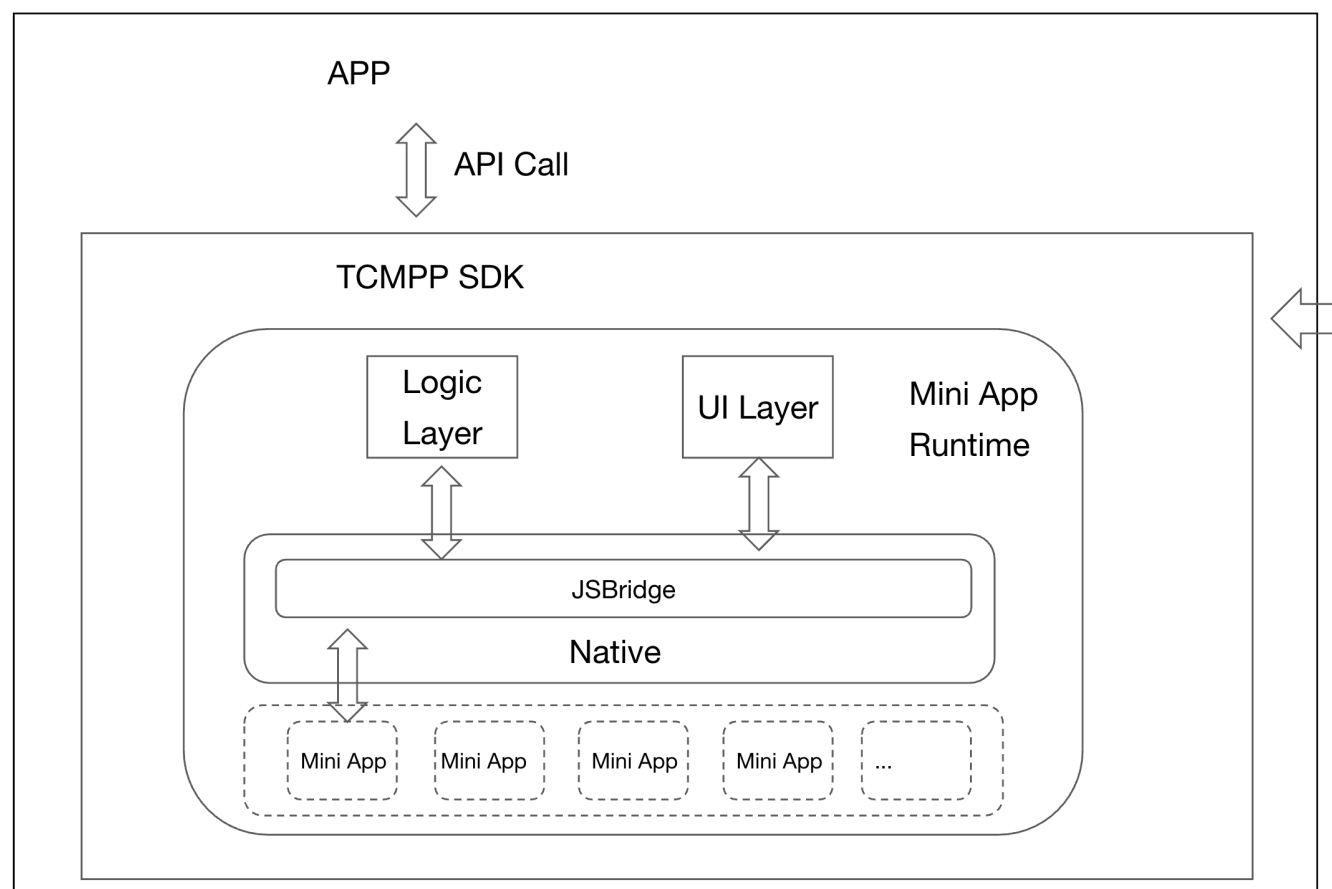


管理控制台：表示 TCMPP 控制台；包含有小程序管理、客户端应用管理、小程序运维管理等能力。

客户端：表示集成 TCMPP 小程序 SDK 的移动客户端应用程序；TCMPP 小程序 SDK 为客户端应用提供小程序的运行时环境。

开发者工具：小程序开发者使用小程序 IDE 进行小程序的开发、调试和版本提交；小程序开发者使用小程序预览助手，在移动客户端进行小程序预览验证。

TCMPP 小程序 SDK 工作原理



小程序 SDK 给客户端应用提供小程序的运行时环境；小程序 SDK 通过和 TCMPP 后台通信，获取小程序信息，并在小程序 SDK 提供的运行时环境中加载和运行小程序。

小程序 SDK 出于减少包大小和减少系统权限及功能控制的考虑，把 SDK 分为了核心库（即 TCMPPSDK）和扩展库（即 TCMPPExtXXX），大部分功能只需要集成核心库就可以实现，并且不需要多余的系统权限申请。只集成核心库对最终 App 安装包的影响在4M左右。

扩展库的相关说明请参考 [扩展组件 SDK](#) 文档。

SDK 快速集成

最近更新时间：2024-07-29 16:47:09

集成样例代码

您可以通过[GitHub](#)获取[Demo](#)地址

前置条件

环境要求

iOS >= 9.0

Xcode >= 10.0

组件依赖

tars

MQQComponents

TMFShark

TMFProfile

SSZipArchive

PromiseObjC

MJRefresh

Masonry

SocketRocket

Brotli

CocoaAsyncSocket

Lame

TMFJSBridge

TMFUploader

集成方式

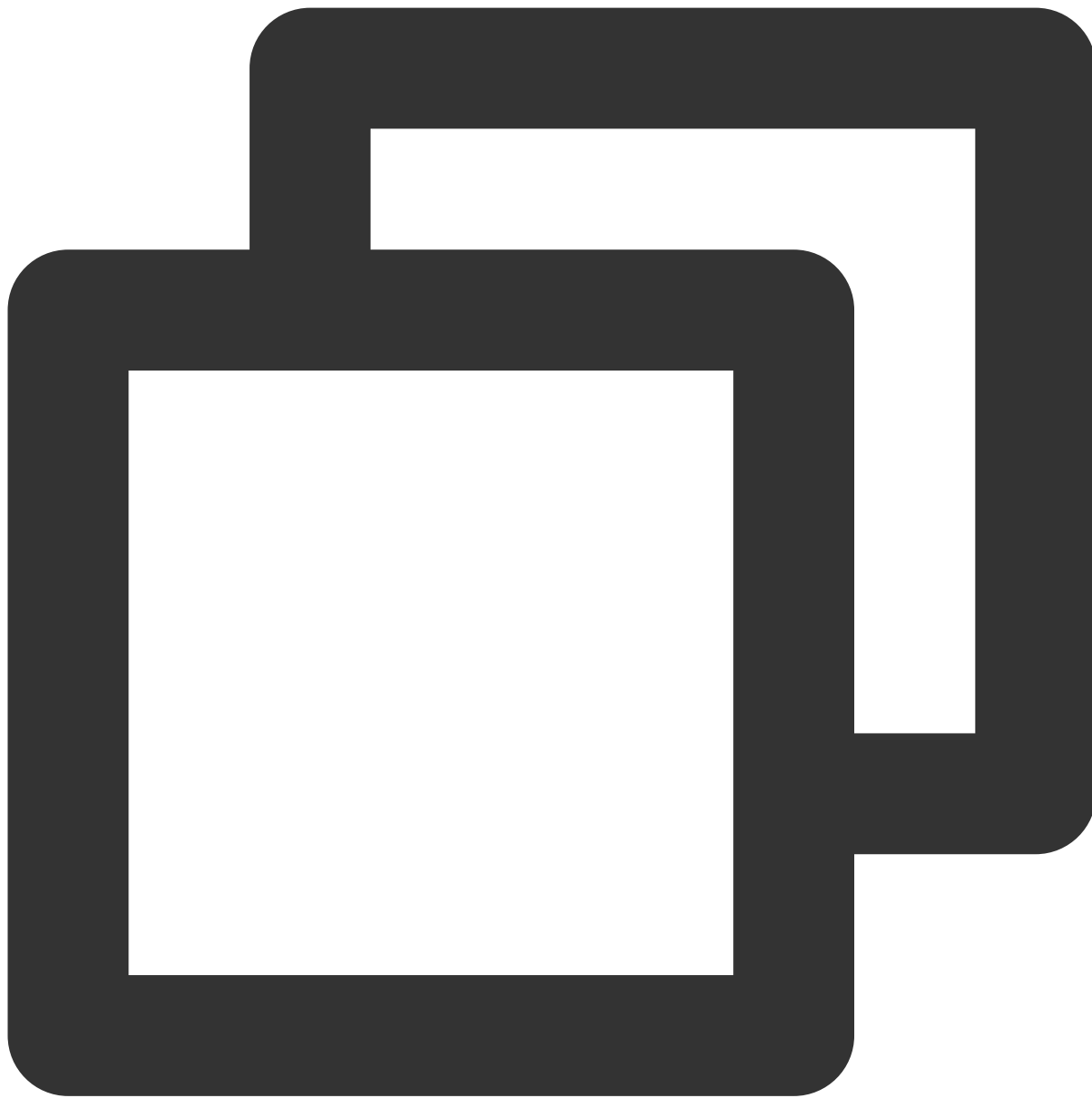
TCMPPSDK 的集成方式有以下2种，可选择其一进行集成：

CocoaPods 集成 SDK

手动集成 SDK

CocoaPods 集成 SDK

1. 在您项目中的 Podfile 文件里添加 TCMPP 源及小程序依赖模块：



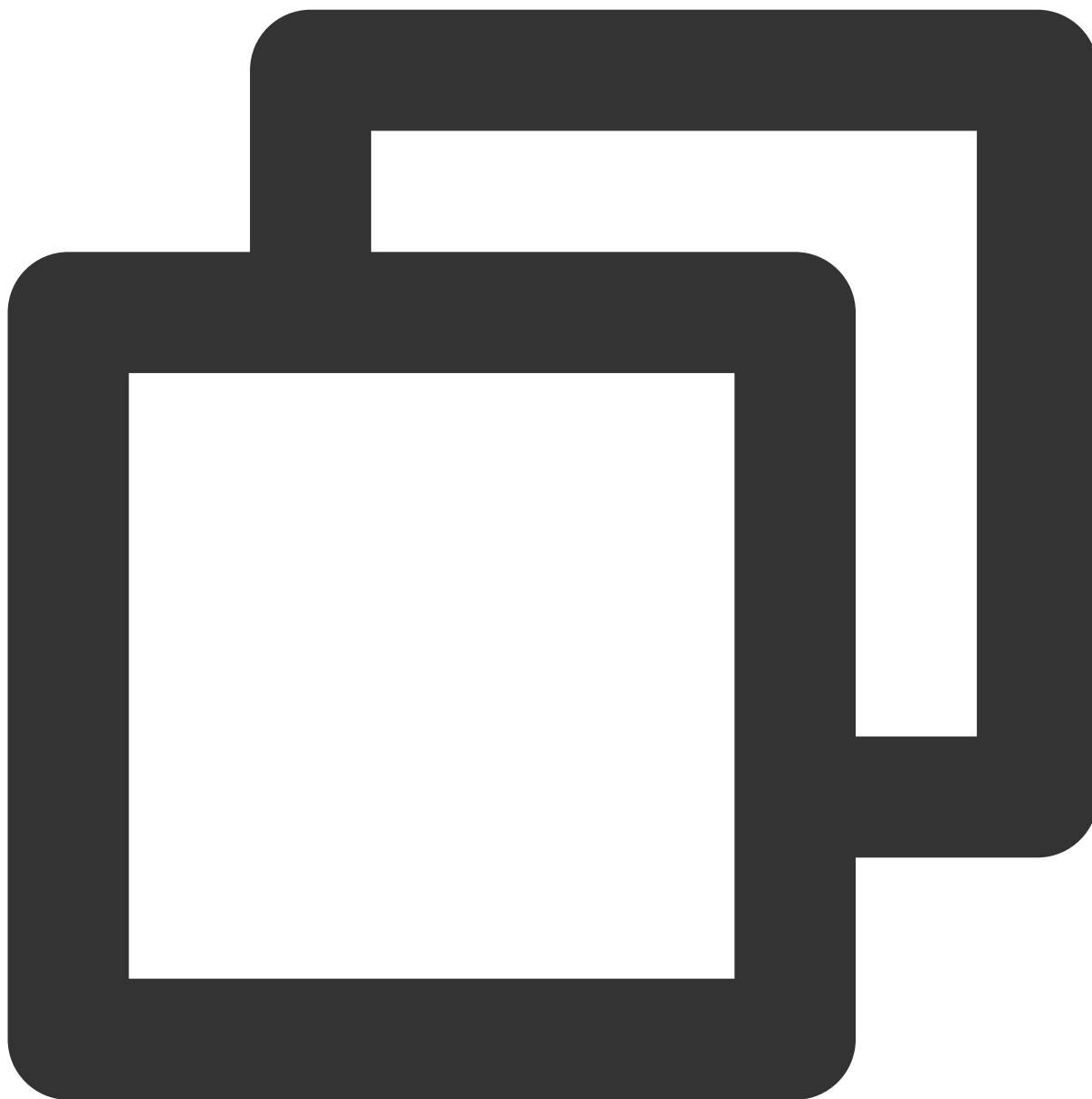
```
#TCMPP Pods 仓库
source 'https://e.coding.net/tcmpp-work/tcmpp/tcmpp-repo.git'

target 'YourTarget' do
  # —— TCMPP ----- #
  pod 'TCMPPSDK'
  pod 'TCMPPExtScanCode'
  pod 'TCMPPExtMedia'
```

```
end
```

其中, `YourTarget` 为您的项目需要引入 `TCMPP` 的 `target` 的名字。

2. Terminal `cd` 到 `Podfile` 文件所在目录, 并执行 `pod install` 进行组件安装。



```
$ pod install
```

说明：

如果报 `Couldn't determine repo type for URL: 'https://e.coding.net/tcmpp-work/tcmpp/tcmpp-repo.git'` 错误, 则需要在执行 `pod install` 前执行 `pod repo add specs https://e.coding.net/tcmpp-`

[work/tcmpp/tcmpp-repo.git](https://github.com/work/tcmpp/tcmpp-repo.git)。

手动集成 SDK

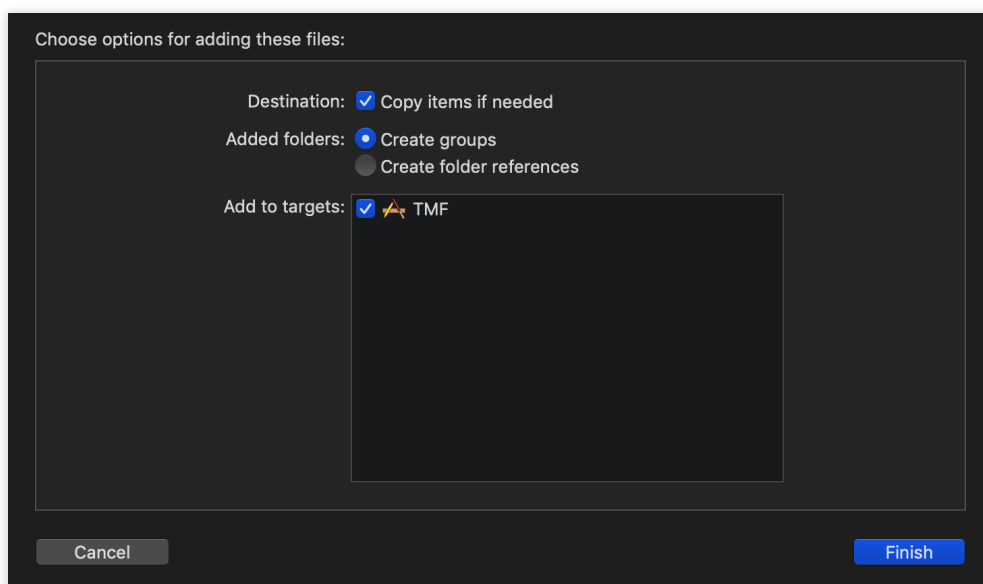
集成样例代码

您可以 [点击下载手动集成 Demo](#)

1. 添加 SDK

将 TCMPPSDK 组件的目录添加到您项目的 Xcode Project 中的合适位置，并选择合适的 target。

您可以把组件的目录从 Finder 直接拖动到 Xcode Project 中，以进行快捷添加。



2. 添加依赖的 SDK

把 TCMPPSDK 依赖的所有组件添加到项目中，依赖的组件列表，请参见前置条件中的 [组件依赖](#)。

3. 添加依赖的系统库

把 TCMPPSDK 依赖的系统库添加到项目中，在 Xcode 中打开 project 设置页，选中相关的 target，单击 **General**，在“Linked Frameworks and Libraries”中进行添加。

4. 系统库依赖如下：

Foundation.framework

CoreTelephony.framework

CFNetwork.framework

Security.framework

SystemConfiguration.framework

CoreService.framework

CoreFoundation.framework

libz.tdb

libc++.tbd

libc.tbd

libz2.tbd

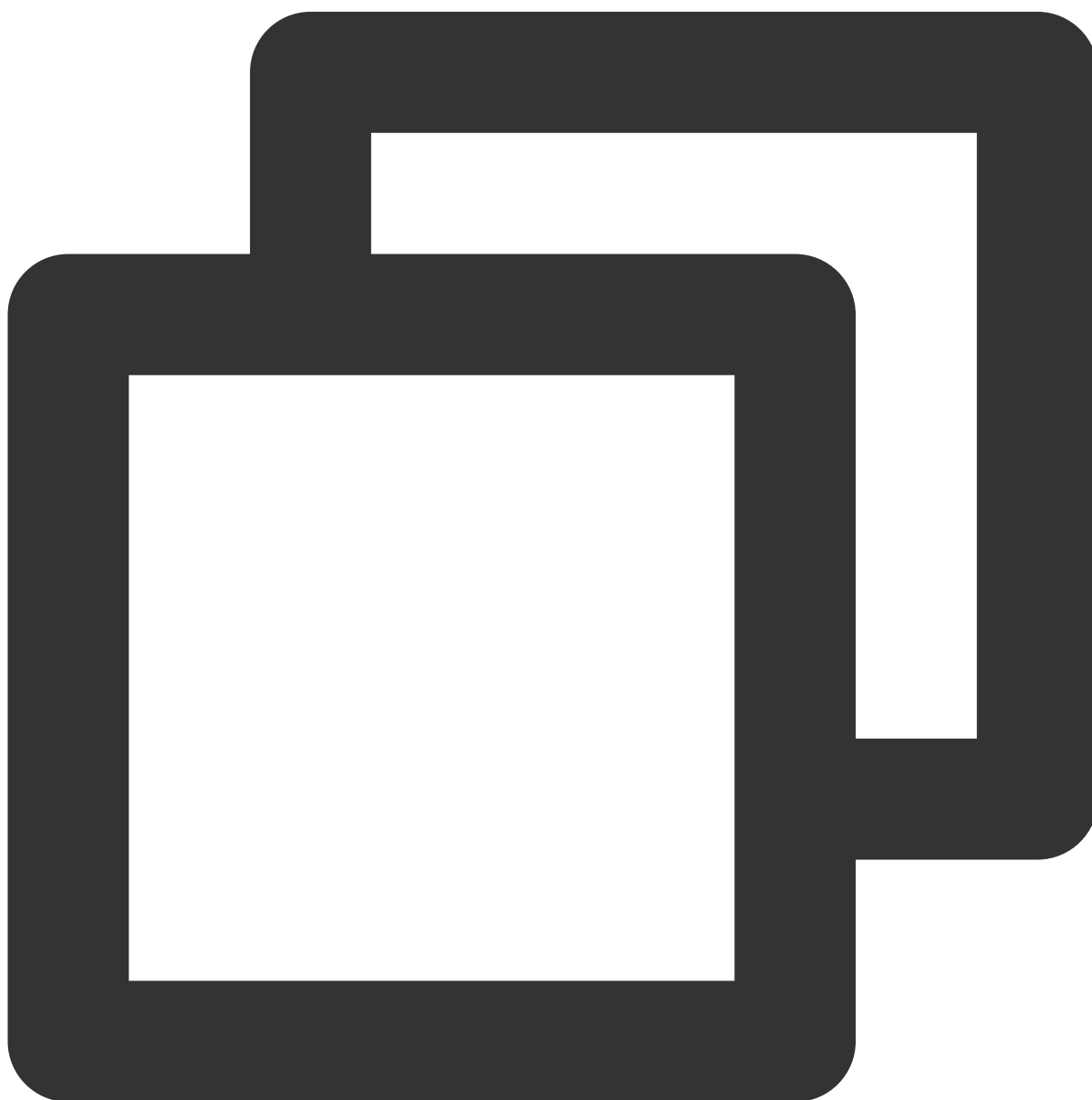
libsqlite3.0.tbd

5. project 设置

添加 TCMPPSDK 后，需要进行相关的 Project 设置。在 Xcode 中打开 Project 设置页，选中相关的 target，进行以下设置：

选择 **Build Settings > Linking > Other Linker Flags**，增加：`-ObjC`。

选择 **Build Settings > Apple Clang - Custom Compiler Flags > Other C Flags**，增加：



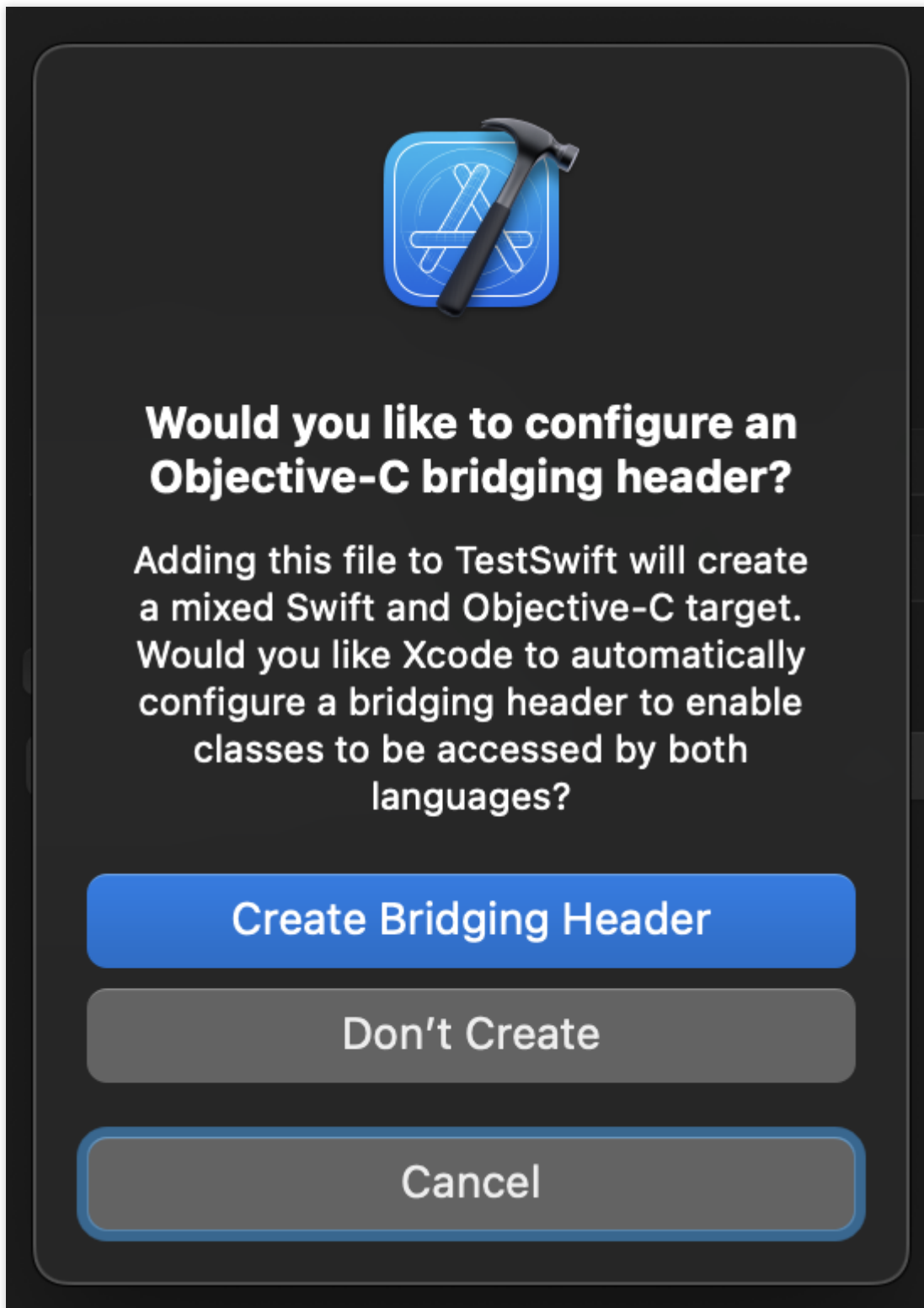
```
-fshort-wchar  
-D__FIXWCHAR__
```

Swift工程集成 SDK

集成样例代码

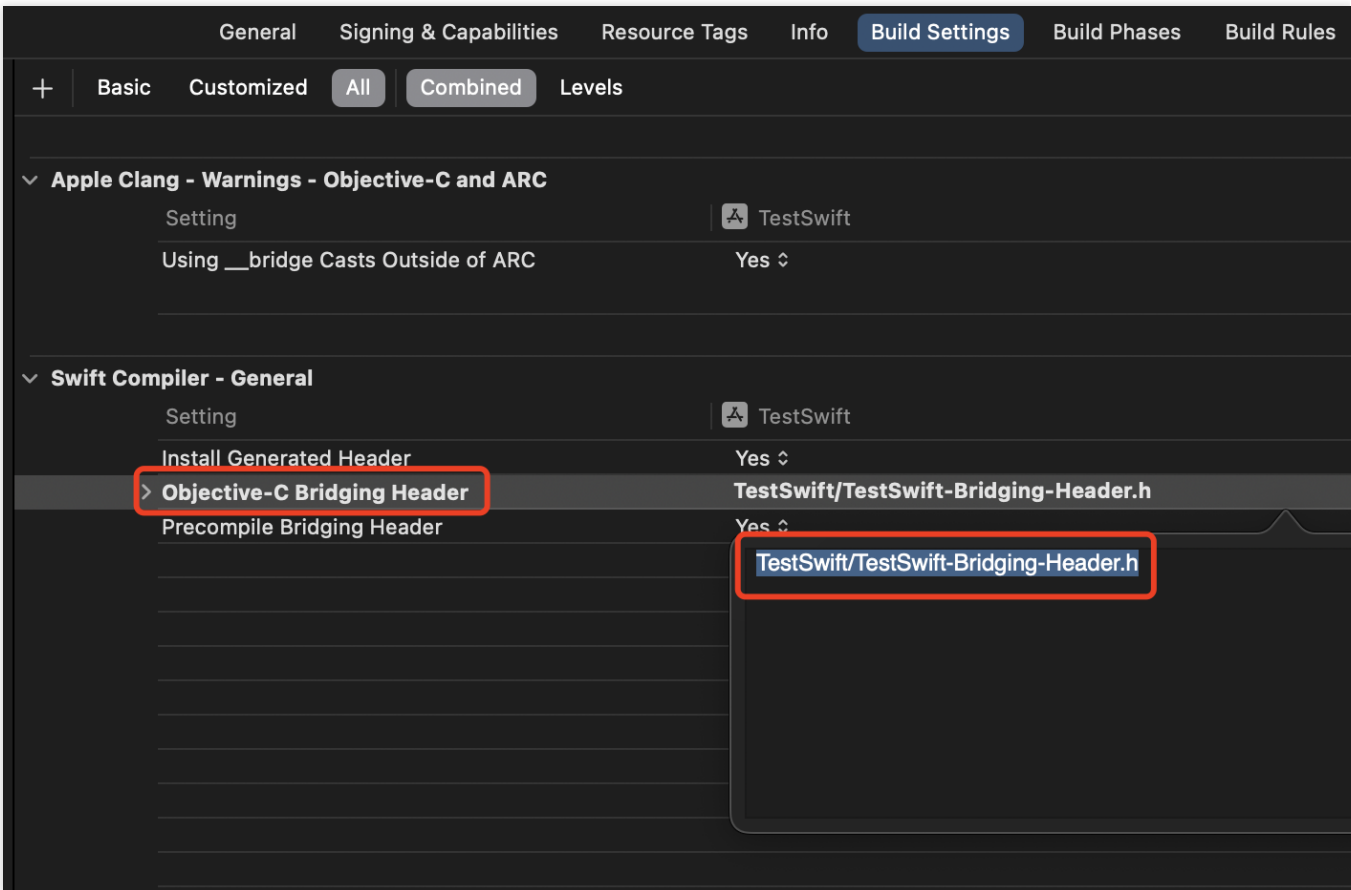
您可以 [点击下载 Swift 工程集成 Demo](#)

第一次引入 OC 文件时，Xcode 会提示是否创建桥接文件，单击创建。

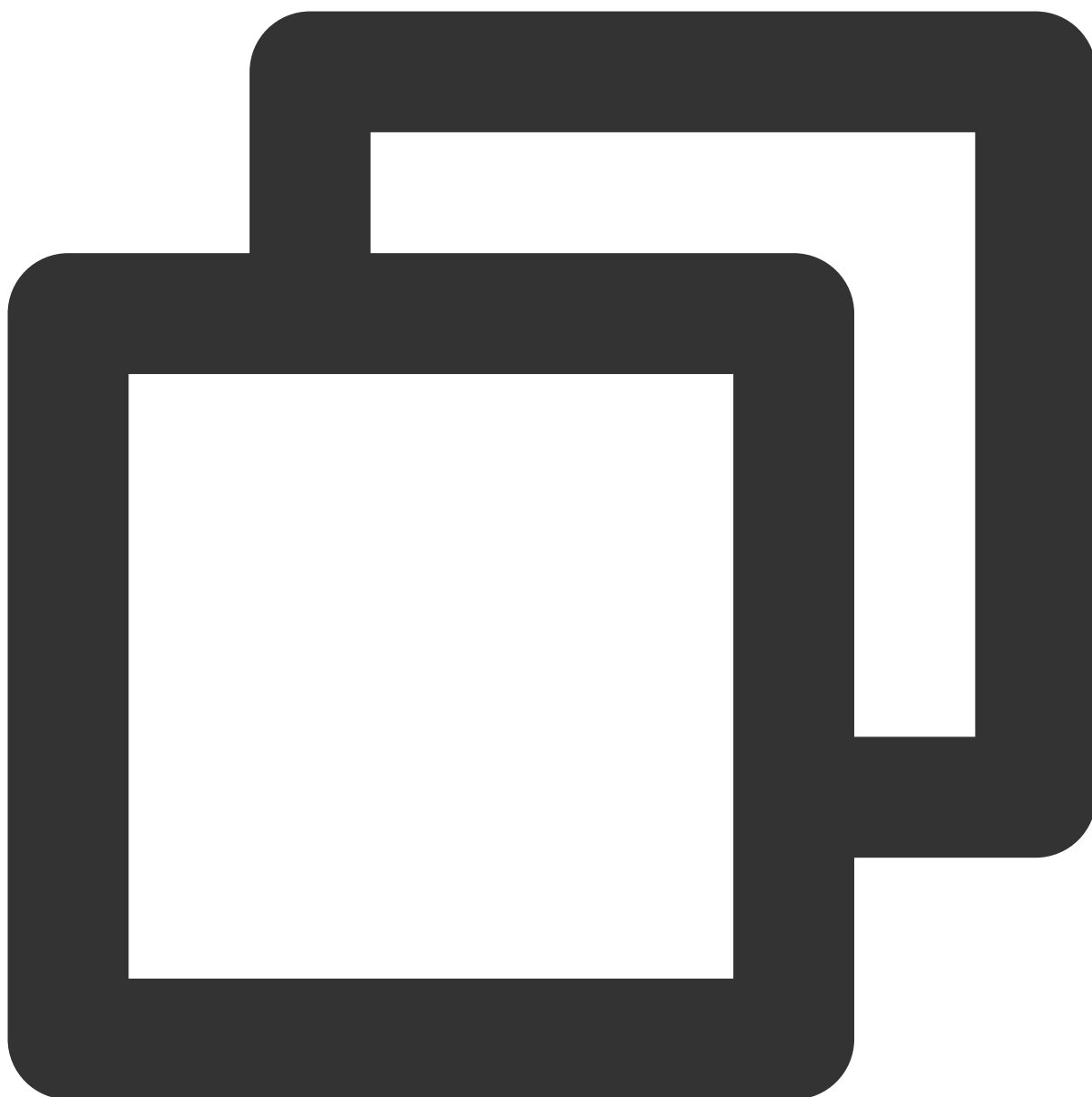


如果你点击了取消，可以手动新建一个 Header，命名为"`{target名-Bridging-Header.h}`"，如"`TestSwift-Bridging-Header.h`"。

并在工程项目的 targets 里的 Build Settings 选项里，找到 Objective-c Bridging Header 选项，设置为创建好的桥接文件。



在该桥接文件里添加代码：

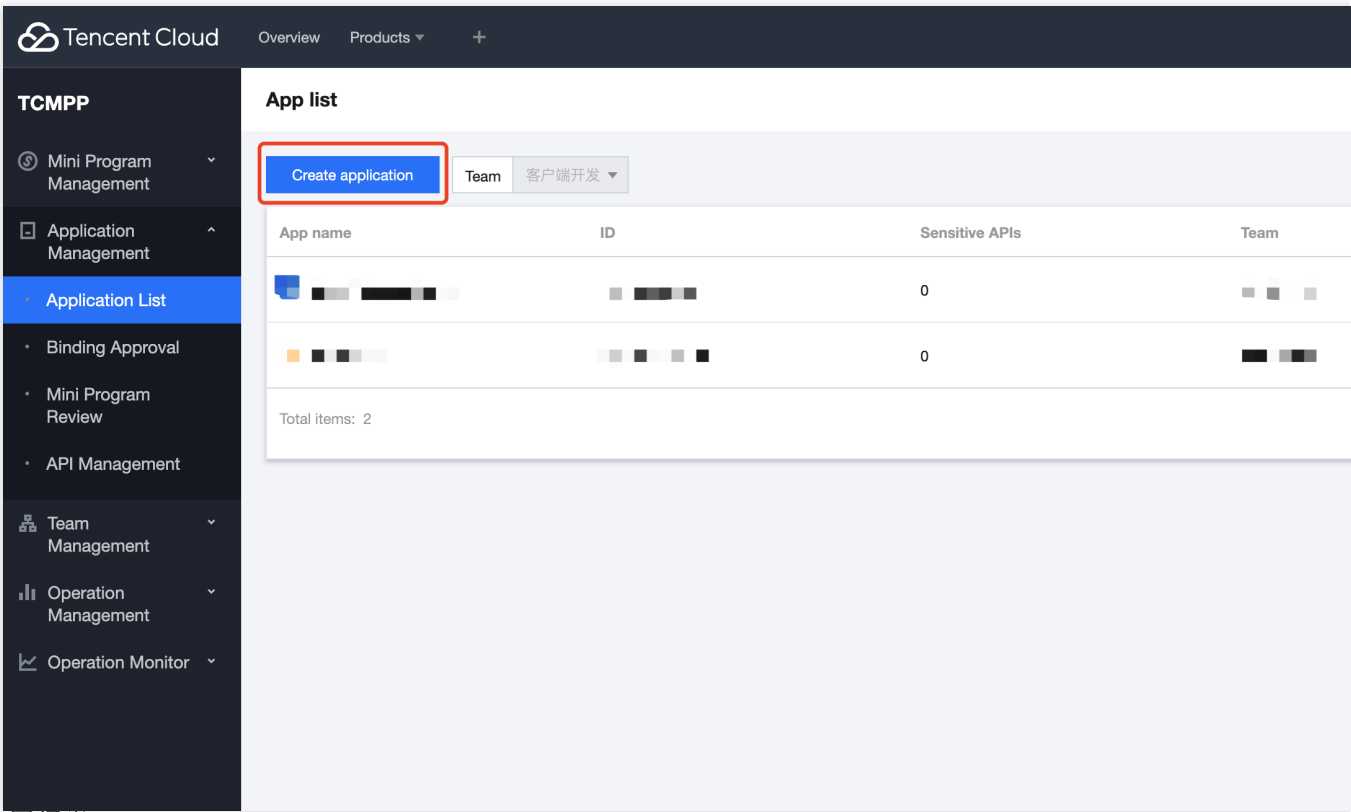


```
#import "TCMPPSDK.h"
```

获取配置文件

小程序 SDK 的初始化依赖于从小程序控制台获取的小程序 SDK 配置文件；在开始集成小程序 SDK 前，需要从小程序控制台获取小程序 SDK 配置文件。

登录控制台后，请单击创建 App：



填写应用信息：

Step 1: Fill in the app information

1 Create application



2 Integrate mini program container

Team *

客户端开发

App name *

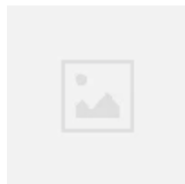
TestApp

Supports 3-64 characters consisting of Chinese characters, uppercase and lowercase letters, numbers, spac

App overview

Please enter the app description.

App icon



No image is selected.

Select image

Please upload a square image in .jpg and .png format with a resolution of 128*128. The image is less than 2 MB. If the icon is not uploaded, the system default icon will be used.

Integration platform *

☒ iOS platform

Bundle ID *

com.testCompany.testApp

App download address

Please enter the app download address

☐ Android Platform

Next Step, Integrate the Mini Program Container

下载配置文件：

注意：

默认下载的配置文件名称为：tcmpp-ios-configurations.json

Step 2: Integrate the mini program container and enable programs.

✓ Create application > 2 Integrate mini program container

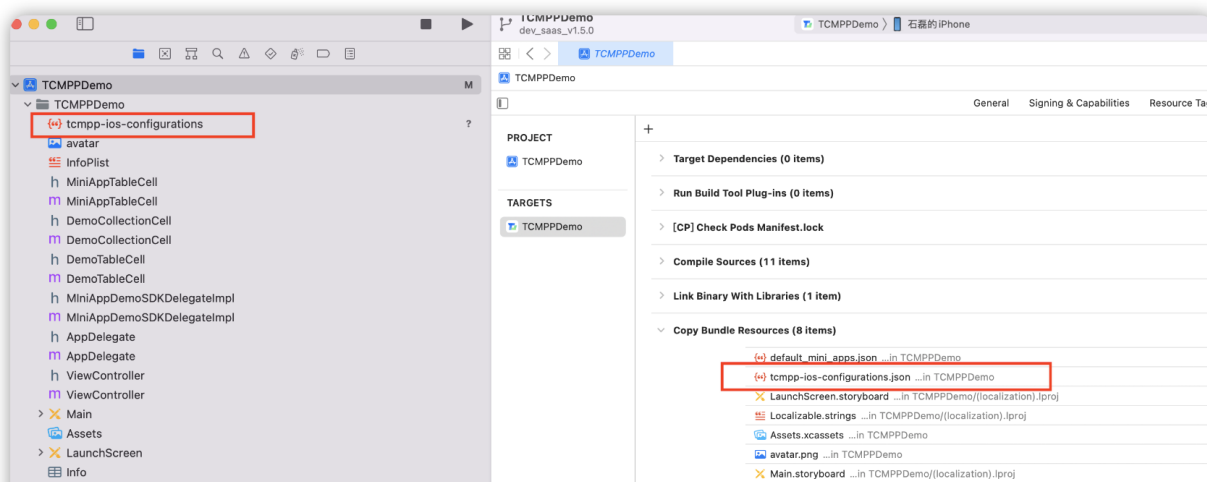
1 Specify the app package name and obtain the configuration file.

Bundle ID com.testCompany.testApp

Configuration file ⓘ

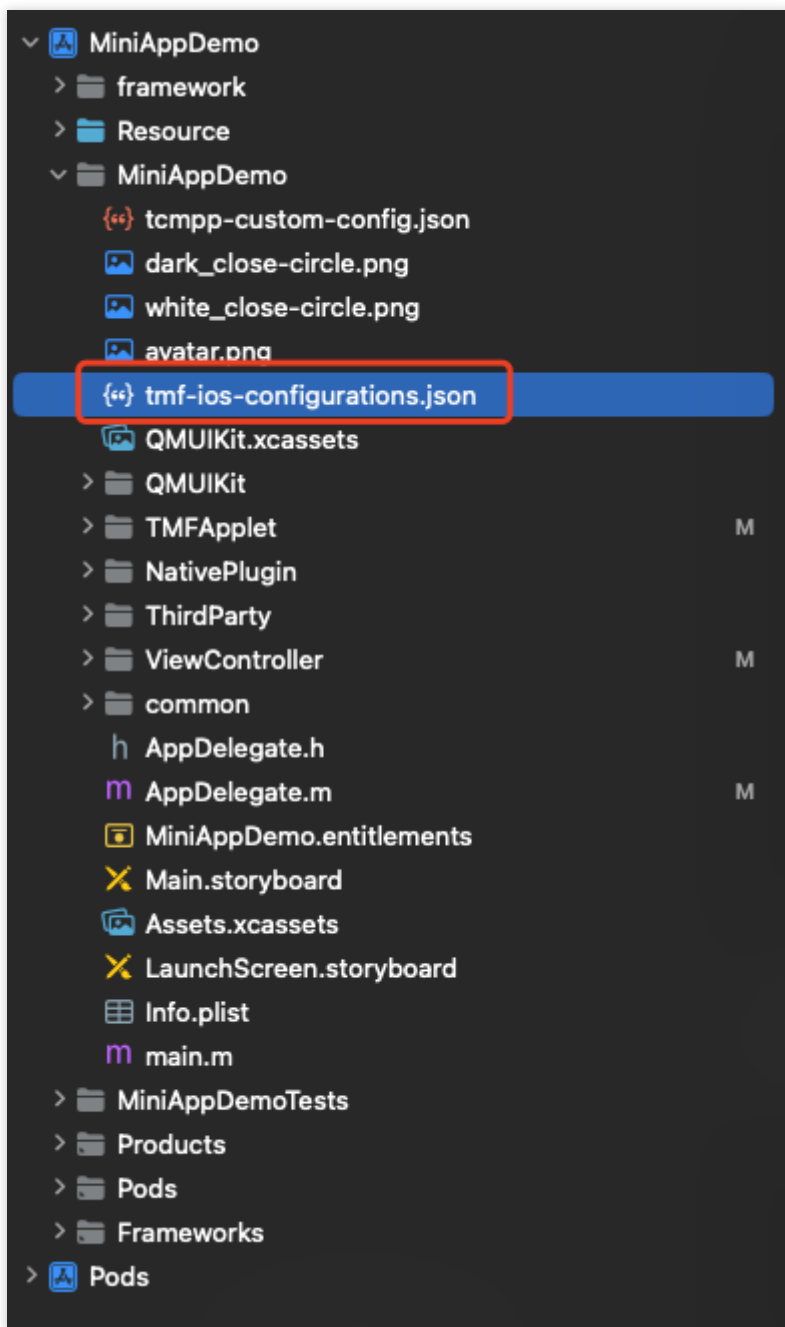
[Download configuration file](#)

2 Incorporate the configuration file into the application and add it to the packaged resources.



添加配置文件到项目

获取配置文件后，需要将配置文件添加到应用程序源码工程中：

**注意：**

iOS 工程的 bundleId 必须与配置文件中 bundleId 保持一致，否则小程序 SDK 在运行时无法通过包名校验，导致 SDK 初始化异常。

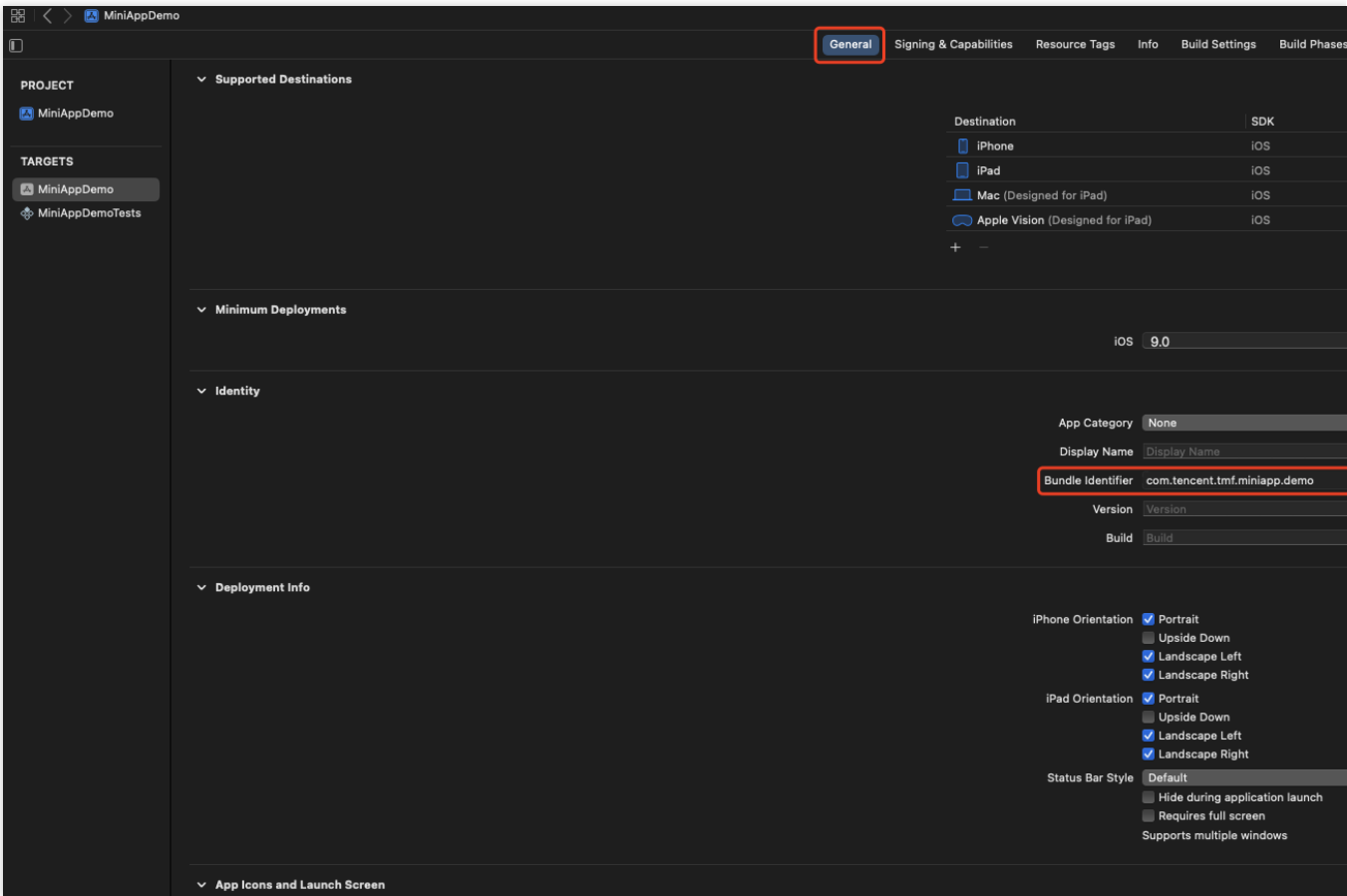
如果不一致，不能直接修改配置文件中的 bundleId 字段，应通过下面两种方式来修正：

修改工程中应用的 bundleId，与配置文件中的 bundleId 一致。

控制台修改应用的 bundleId，与工程中应用的 bundleId 保持一致，并重新下载配置文件。

配置文件中的 packageName 字段，需要和当前项目的包名一致。

```
{
  "customId": "T1701314211HMOVJP",
  "productId": "4007",
  "packageName": "",
  "bundleId": "com.tencent.tcmpp.demo",
  "material":
    "01+cpJbB8WktGpiif0wnkta7qerkNi/MAGHHx9KF3GzPVX1tn/Dp
    rizbZBRhxdCopWC16R1U/zla2oa3PK3ayWibFCPjFLO8RZJpHtnKw
    0bHCXBK5oQUR9fs2AARuY4kSIR4gLLb7ZnfjUVdViHAHPmxPr5IJG
    iv904cSIfelp2RdvEA=1fd53748f1db282e",
  "shark": {
    "httpUrl": "https://api.tcmpp.tmfcloud.com",
    "tcpHost": "api.tcmpp.tmfcloud.com",
    "tcpPort": 0,
    "appKey": "app-azosmbqqjn",
    "symEnc": 2,
    "asymEnc": 1
  },
  "feedback": "",
  "qapmUrl": "http://ww.com"
}
```



增加权限设置

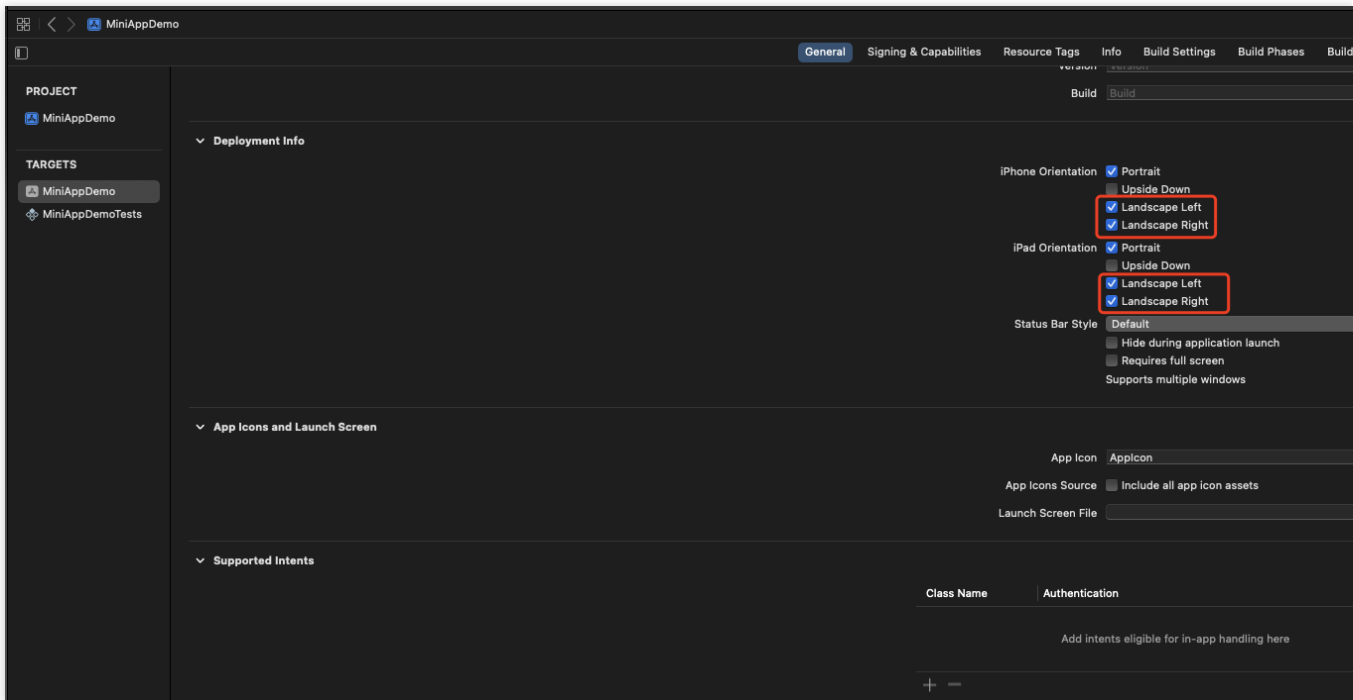
如果您只集成了 TCMPP 的核心库，即 TCMPPSDK，需要在 info.plist 文件中添加权限申请信息。其中核心 SDK(FinApplet) 涉及的权限：

权限名称	权限对应的 KEY	权限涉及的 API
相册写入	NSPhotoLibraryAddUsageDescription	saveImageToPhotosAlbum、saveVideoToPhotosAlbum
相机	NSCameraUsageDescription	CameraContext(Camera组件)
麦克风	NSMicrophoneUsageDescription	CameraContext(Camera组件)

如果您需要使用扩展 SDK，那么就要在项目的 info.plist 文件中添加对应的权限申请信息。可以查看 [扩展组件 SDK](#) 相关文档。

设置设备支持横屏

TCMPP SDK 中的 loading 页面，video 组件等都支持横屏效果，但前提是 App 工程支持横屏。
请在 Xcode 设置中勾选以下选项：

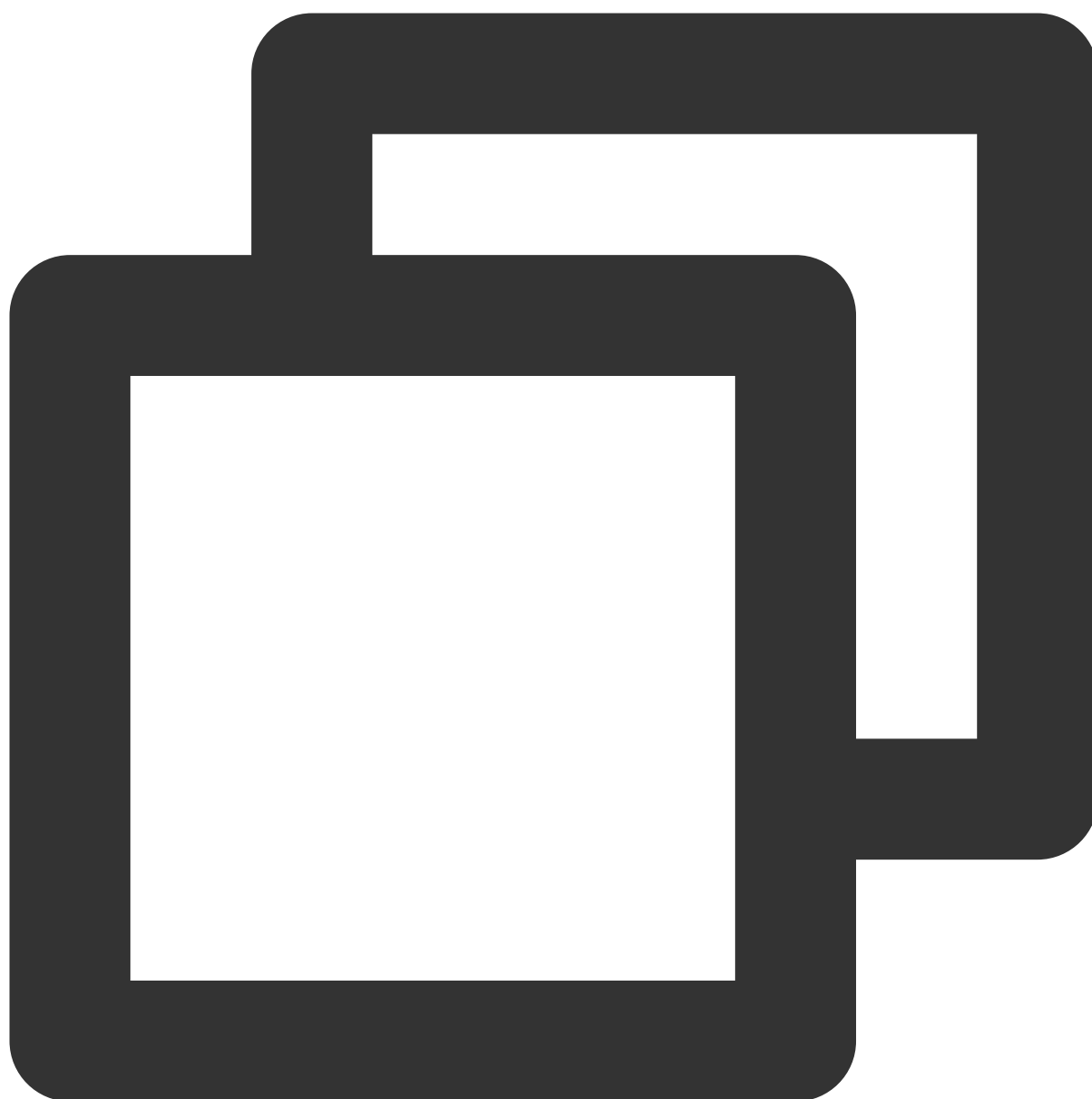


注意：

如果不勾选横屏，那么 SDK 中涉及到横屏的功能就没有效果。

引入头文件

把 TMFMiniAppSDKManager.h 和 TMFAppletConfigManager.h 引入到 AppDelegate 中。



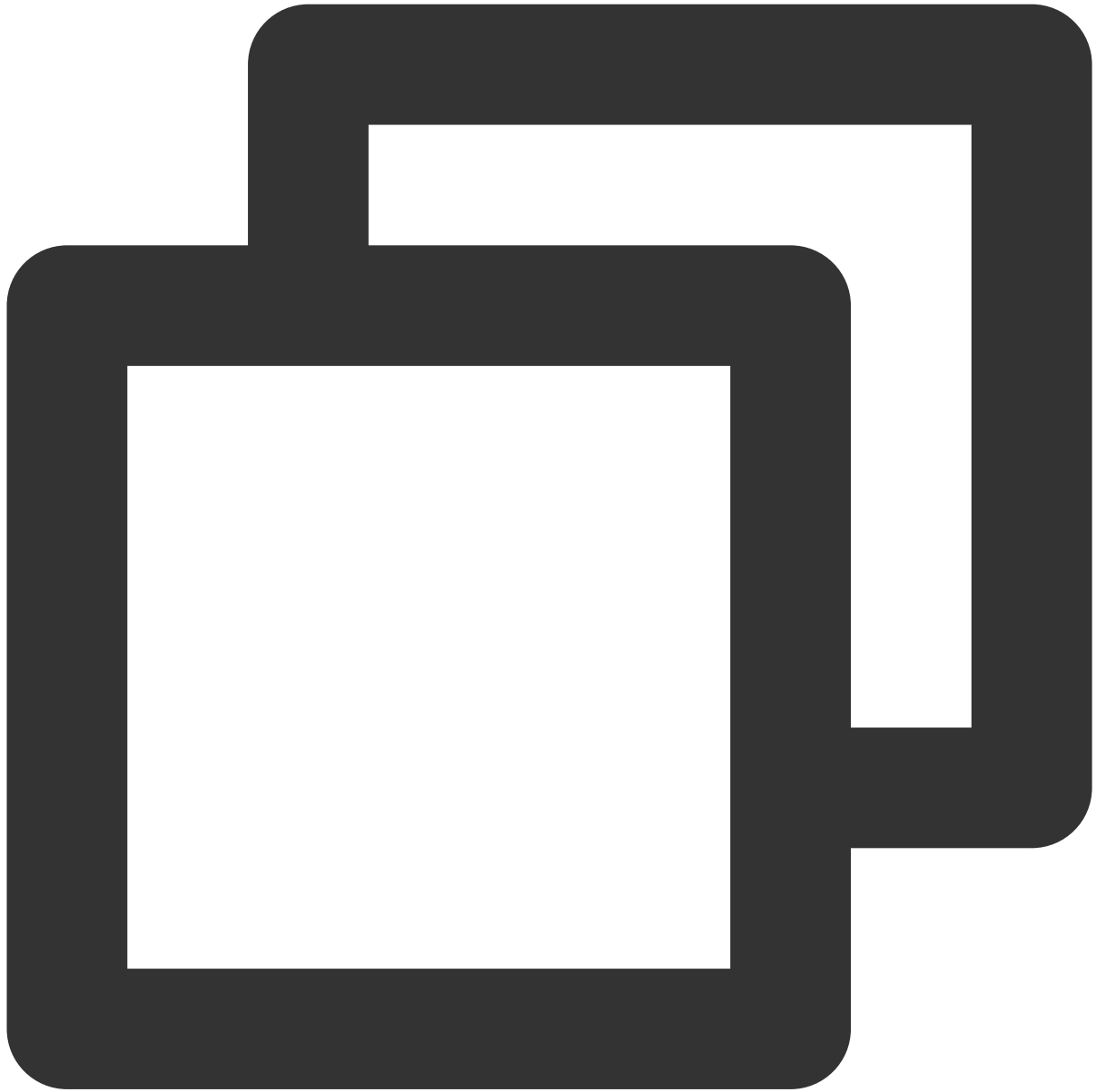
```
//TCMPP  
#import "TCMPPSDK.h"
```

配置信息设置

根据配置文件初始化 `TMA ServerConfig` 对象，并使用 `TMA ServerConfig` 初始化 TCMPP 小程序引擎。

SDK 可以支持直接进行引擎初始化，提前准备网络链接，并更新基础库信息和配置信息，加速后小程序加载，也可以支持在需要的时候进行初始化。

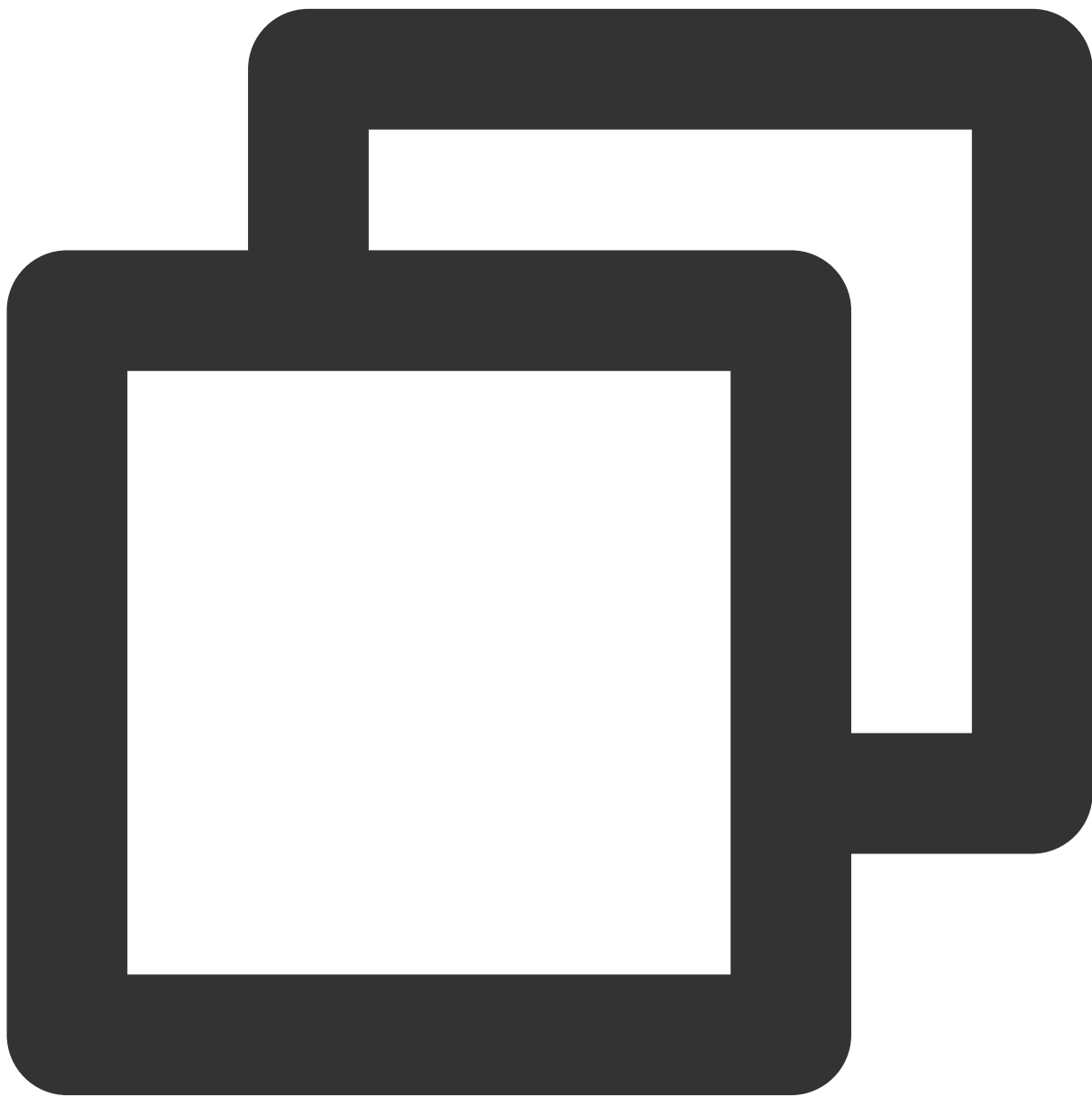
参考代码：



```
//配置使用环境
NSString *filePath = [[NSBundle mainBundle] pathForResource:@"tcmp-ios-configurat
if(filePath) {
    TMA ServerConfig *config = [[TMA ServerConfig alloc] initWithFile:filePath];
    //直接初始化
    [[TMFMiniAppSDKManager sharedInstance] setConfiguration:config];
}
```

其它初始化动作

使用者可根据需要，设置开放接口实现实例。如果需要集成扩展模块时，初始化扩展接口准备。

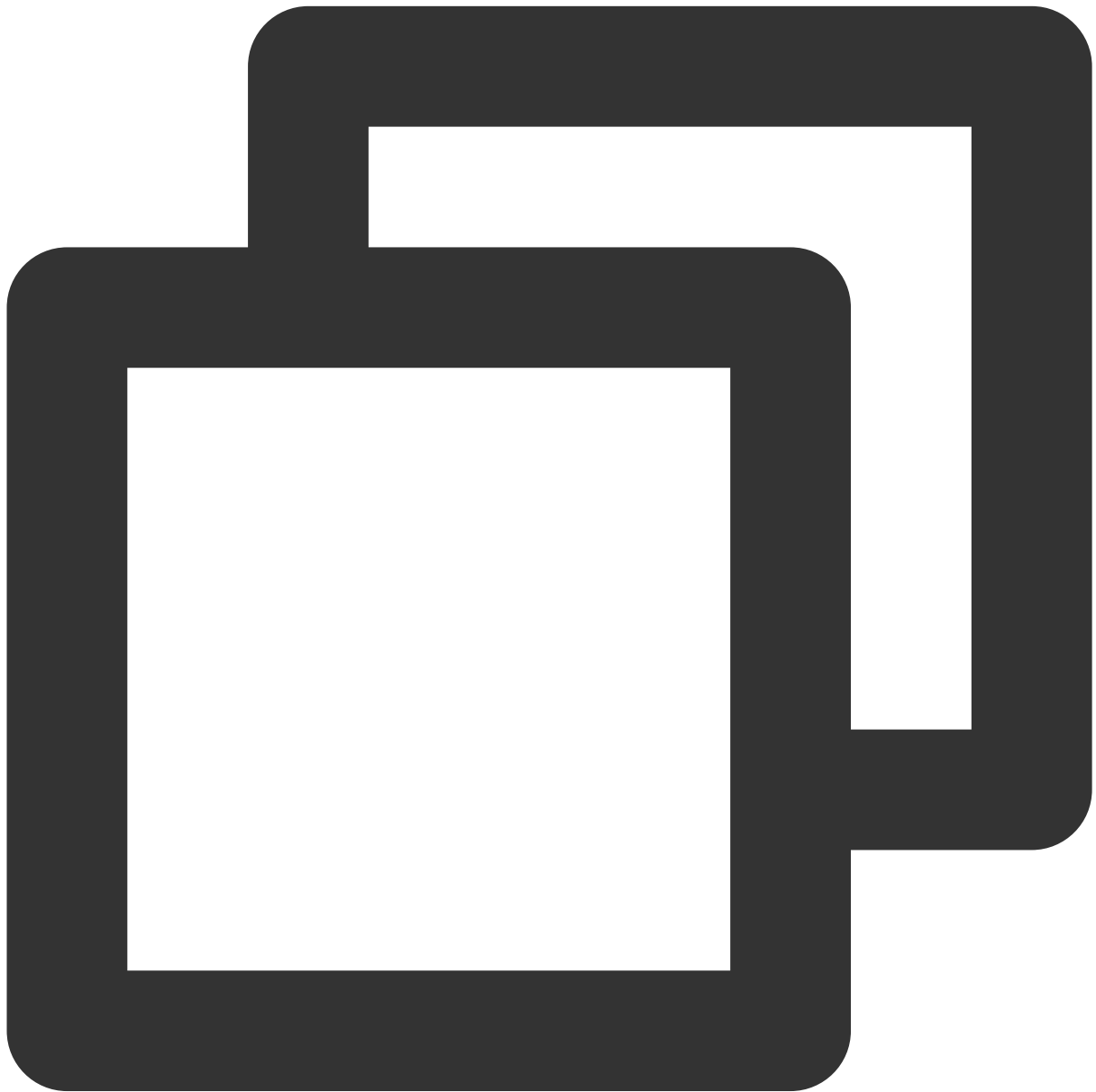


```
//设置小程序引擎代理类实现
[TMFMiniAppSDKManager sharedInstance].miniAppSdkDelegate = [MIniAppDemoSDKDelegateI
```

MlniAppDemoSDKDelegatImpl必须实现**TMFMiniAppSDKDelegate**协议，可以参考 [定制 SDK 能力](#) 和 Demo 工程中的 MlniAppDemoSDKDelegatImpl 文件。

打开小程序

可以通过 AppId 直接调用 API 打开小程序。



```
[[TMFMiniAppSDKManager sharedInstance] startUpMiniAppWithAppID:appId parentVC:self  
NSLog(@"open applet error:%@",error);
```

```
}];
```

SDK 集成常见问题

最近更新时间：2024-07-24 09:57:23

XCode 15 以上版本编译后，在 iOS 12 及以下设备上出现崩溃问题

在项目编译选项中添加 `-Wl,-ld_classic` 配置。



Xcode 15 RC Release Notes

Linking



New Features

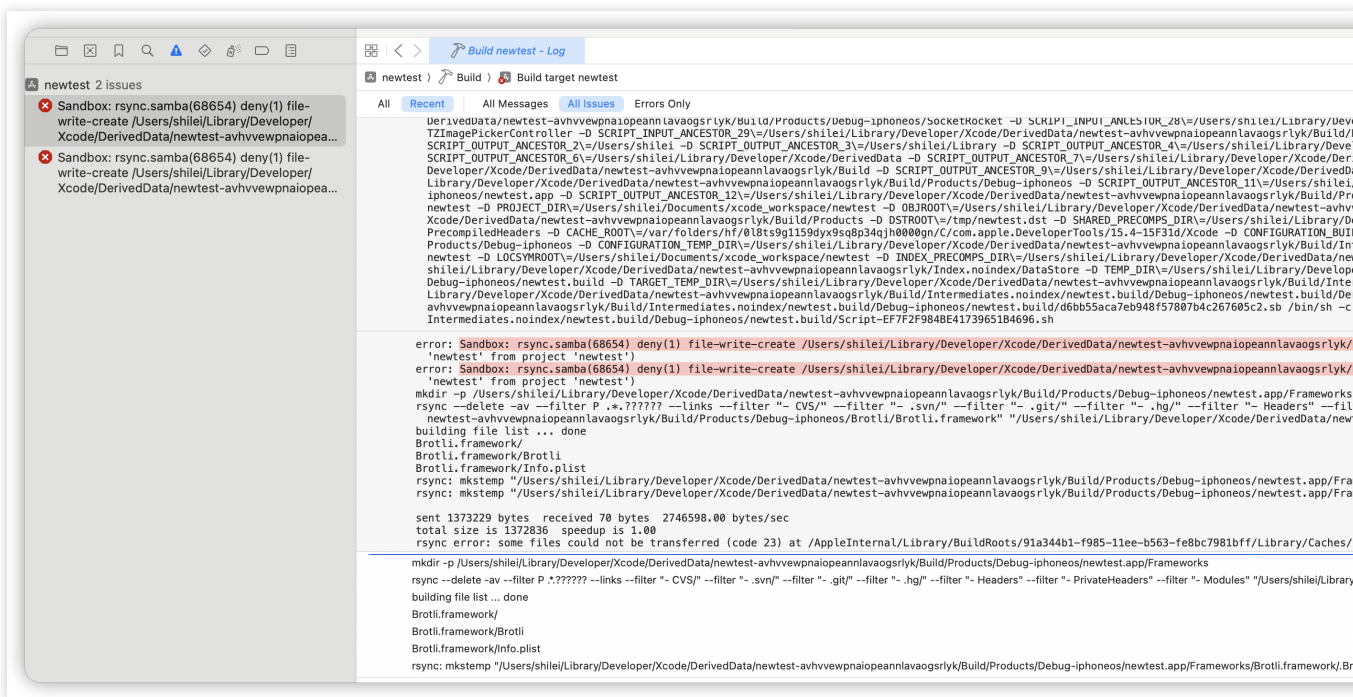
A new linker has been written to significantly speed up static linking. It's the default for all macOS, iOS, tvOS and visionOS binaries and using the "Mergeable Libraries" feature. The classic linker can still be explicitly requested using `-ld64`, and will be removed in a future release (108915312)

Known Issues

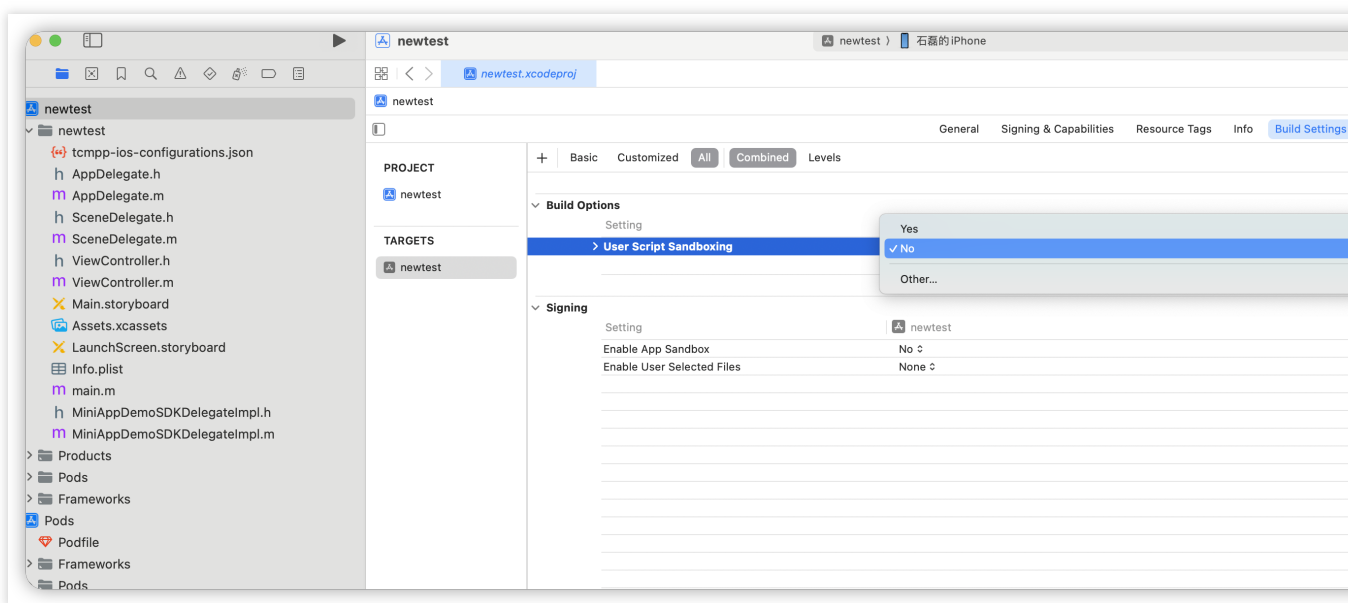
Binaries using symbols with a weak definition crash at runtime on iOS 14/macOS 12 or older. This impacts primarily C++ projects due to extensive use of weak symbols. (114813650) ([FB13097713](#))

Workaround: Bump the minimum deployment target to iOS 15, macOS 12, watchOS 8 or tvOS 15, or add `-Wl,-ld_classic` to the `OTHER_LDFLAGS` build setting.

Xcode15新建项目并且引入pod库后，编译报错 Sandbox：rsync.samba deny(1) file-write-create XXX



解决办法：Build Settings 搜索 sandbox，把 Build Options 中的 User Script Sandboxing 改为 NO。



Apple M 系列芯片电脑模拟器运行异常

当前 TCMPP SDK 版本对于 M 系列芯片电脑模拟器上运行需要通过 Rosetta 运行。

xcode14之前：

我们可以右键 xcode > 显示简介 > 勾选使用 Rosetta 打开，这样就是在模拟器上运行了。

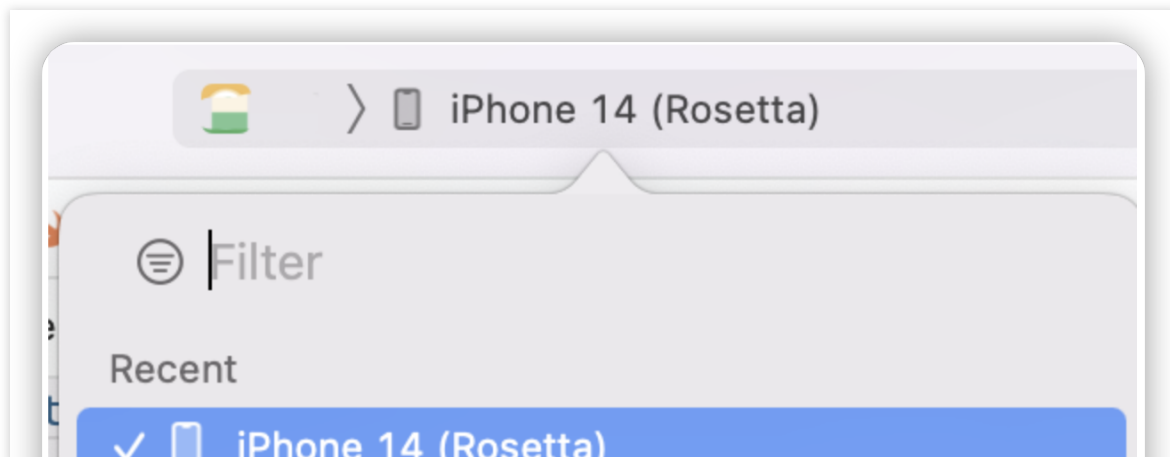
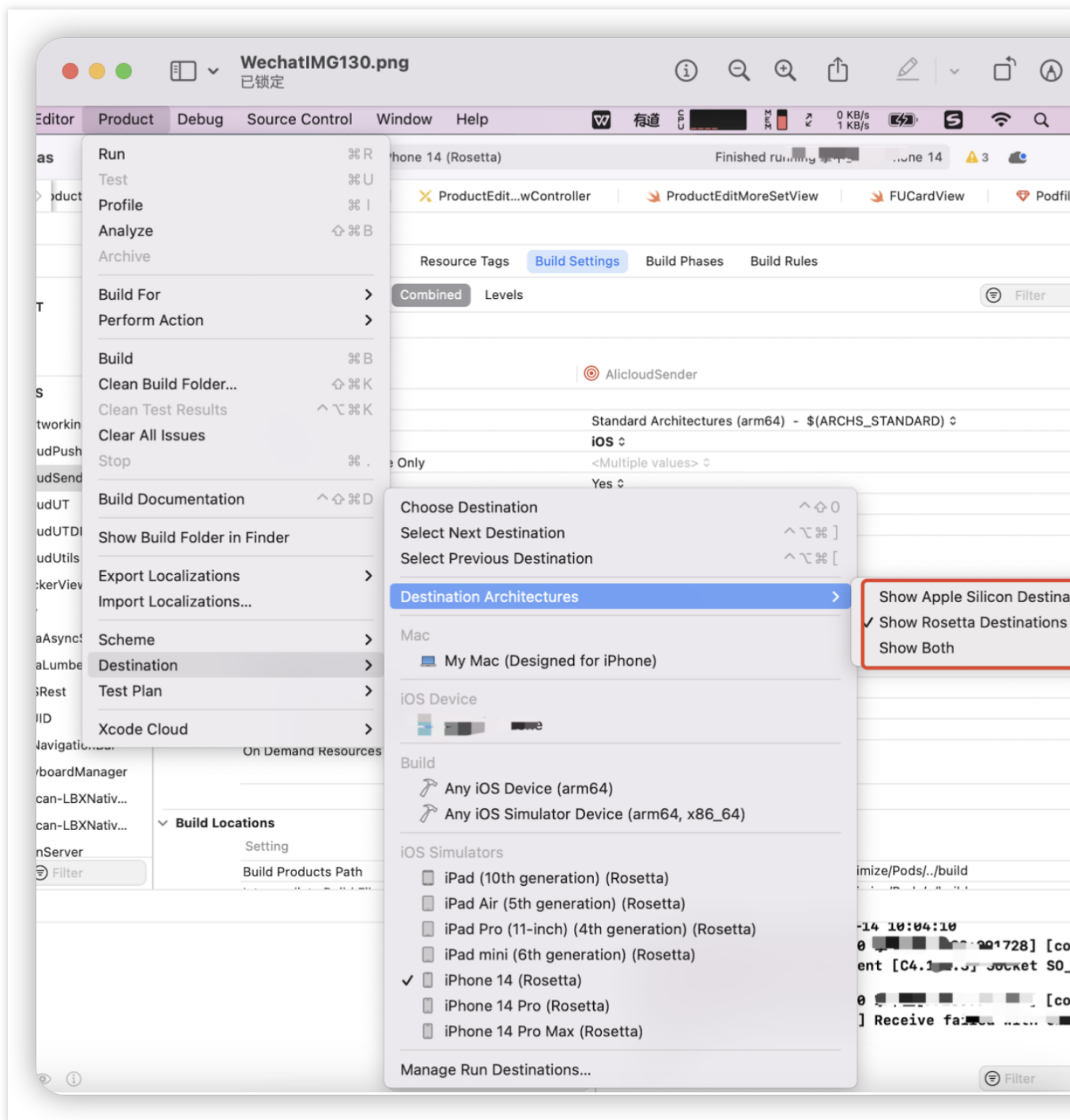


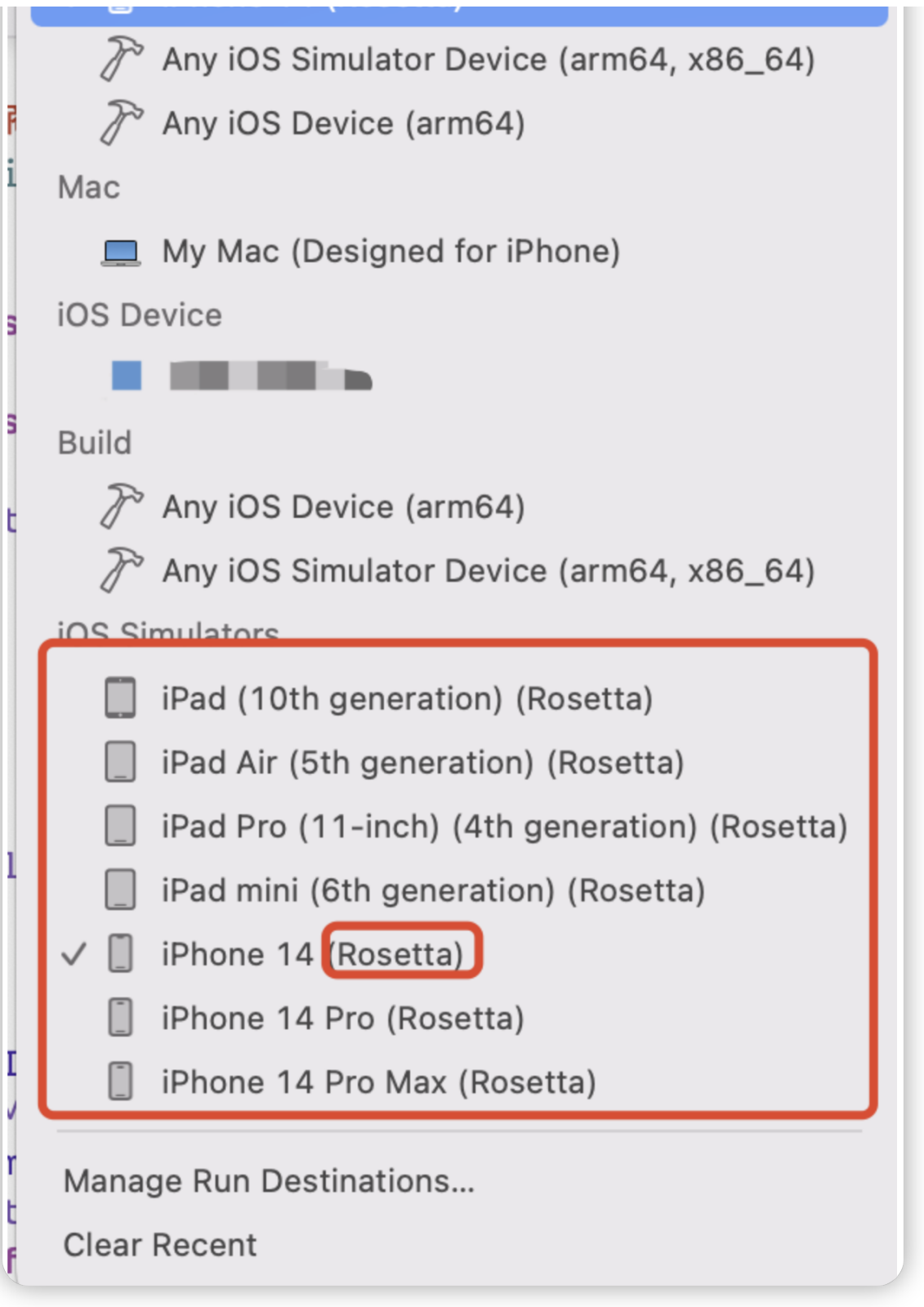


xcode14后：

xcode 打开项目 > Product > Destination > Destination Architectures 可以选择用哪种模式的模拟器打开。

我们选择（Rosetta）结尾的模拟器即可。





iOS API

小程序管理 API

搜索小程序

最近更新时间：2024-07-03 18:00:54

小程序 SDK 提供线上小程序搜索的接口，可以通过此接口对小程序按关键字和分类进行搜索。

说明：

`firstType` 和 `secondType` 用于指定小程序搜索的一级分类和二级分类信息；

`completion` 参数用于获取小程序的搜索结果。

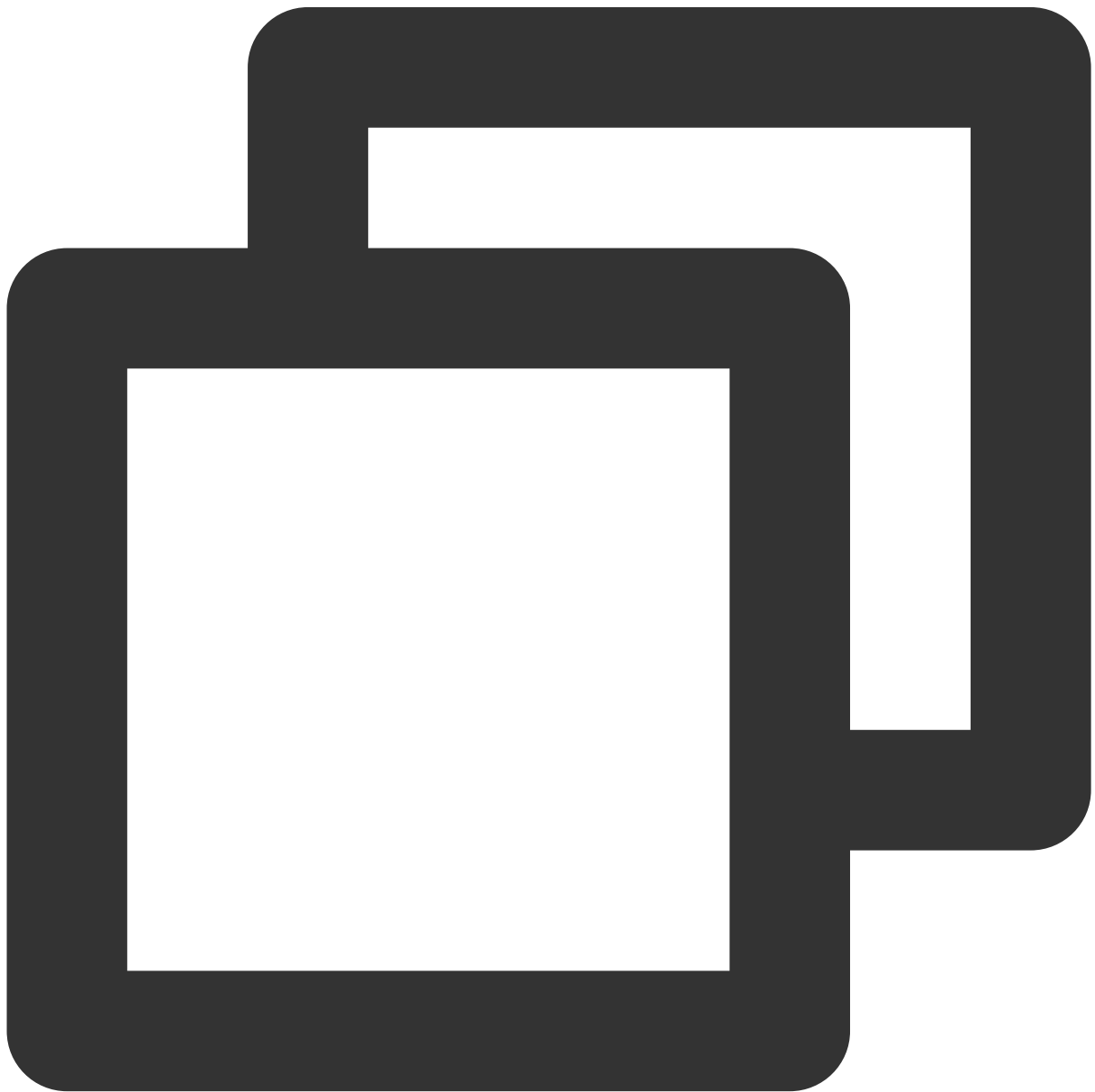


```
// 搜索小程序
// Search mini program
// @param name 搜索名称关键词 - search name keyword
// @param completion 搜索结果 - search results
- (void)searchAppletsWithName:(NSString *)name
                        completion:(void (^)(NSArray<TMFAppletSearchInfo *> * _Nullable))completion {
    // 搜索小程序
    // Search mini program
    // @param name 搜索名称关键词 - search name keyword
    // @param firstType 一级分类名称 - first level classification name
```

```
// @param secondType 二级分类名称 - second level classification name
// @param completion 搜索结果 - search results
- (void)searchAppletsWithName:(NSString * _Nullable )name
                        firstType:(NSString * _Nullable)firstType
                        secondType:(NSString * _Nullable)secondType
                        completion:(void (^) (NSArray<TMFAppletSearchInfo *> * _Nullable,
```

按关键字搜索

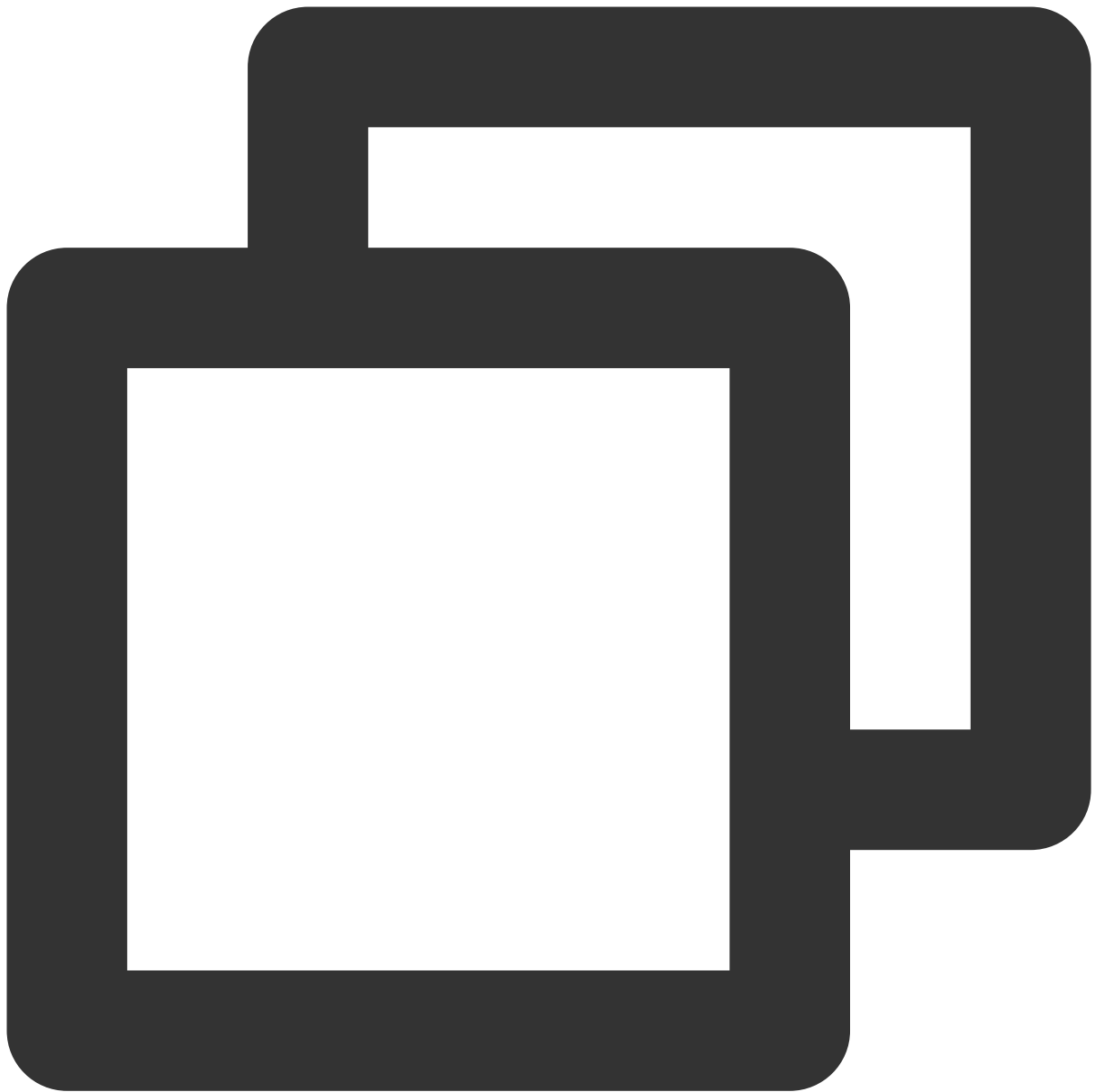
示例代码：



```
[[TMFMiniAppSDKManager sharedInstance] searchAppletsWithName:searchString completion:^(NSArray<TMFMiniApp*> *applets, NSError *error) {  
    if (error) {  
        //搜索失败，或列表为空  
    } else {  
        //搜索成功，列表不为空  
    }  
}];
```

按单分类搜索

示例代码：



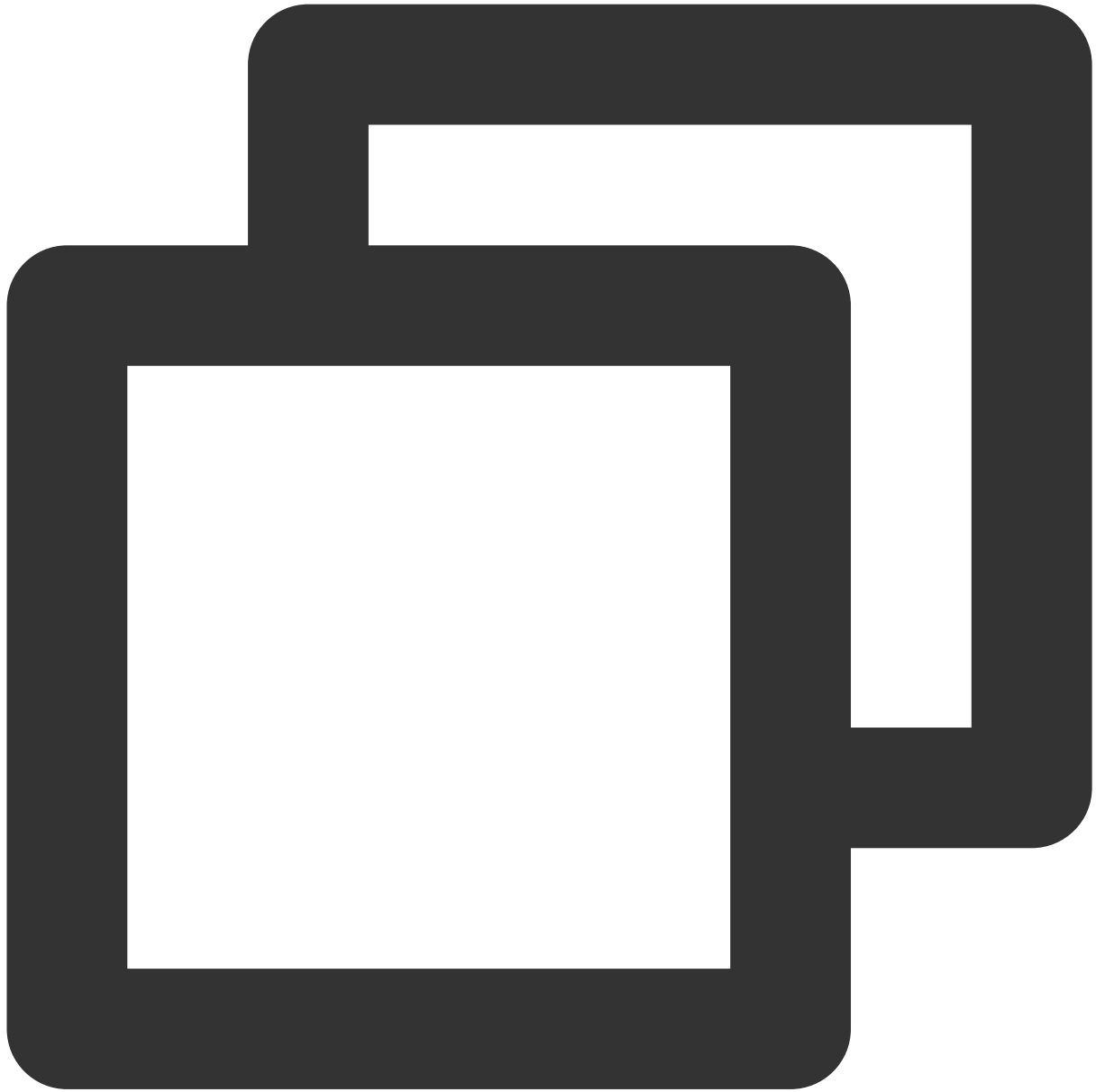
```
[[TMFMiniAppSDKManager sharedInstance] searchAppletsWithName:nil firstType:@"firstT
    if (error) {
        //搜索失败，或列表为空
    } else {
        //搜索成功，列表不为空
    }
}];
```

双分类搜索

说明：

双分类搜索的结果为两个分类的交集。

示例代码：



```
[[TMFMiniAppSDKManager sharedInstance] searchAppletsWithName:nil firstType:@"firstT
    if (error) {
        //搜索失败，或列表为空
    } else {
        //搜索成功，列表不为空
    }
}];
```


打开小程序

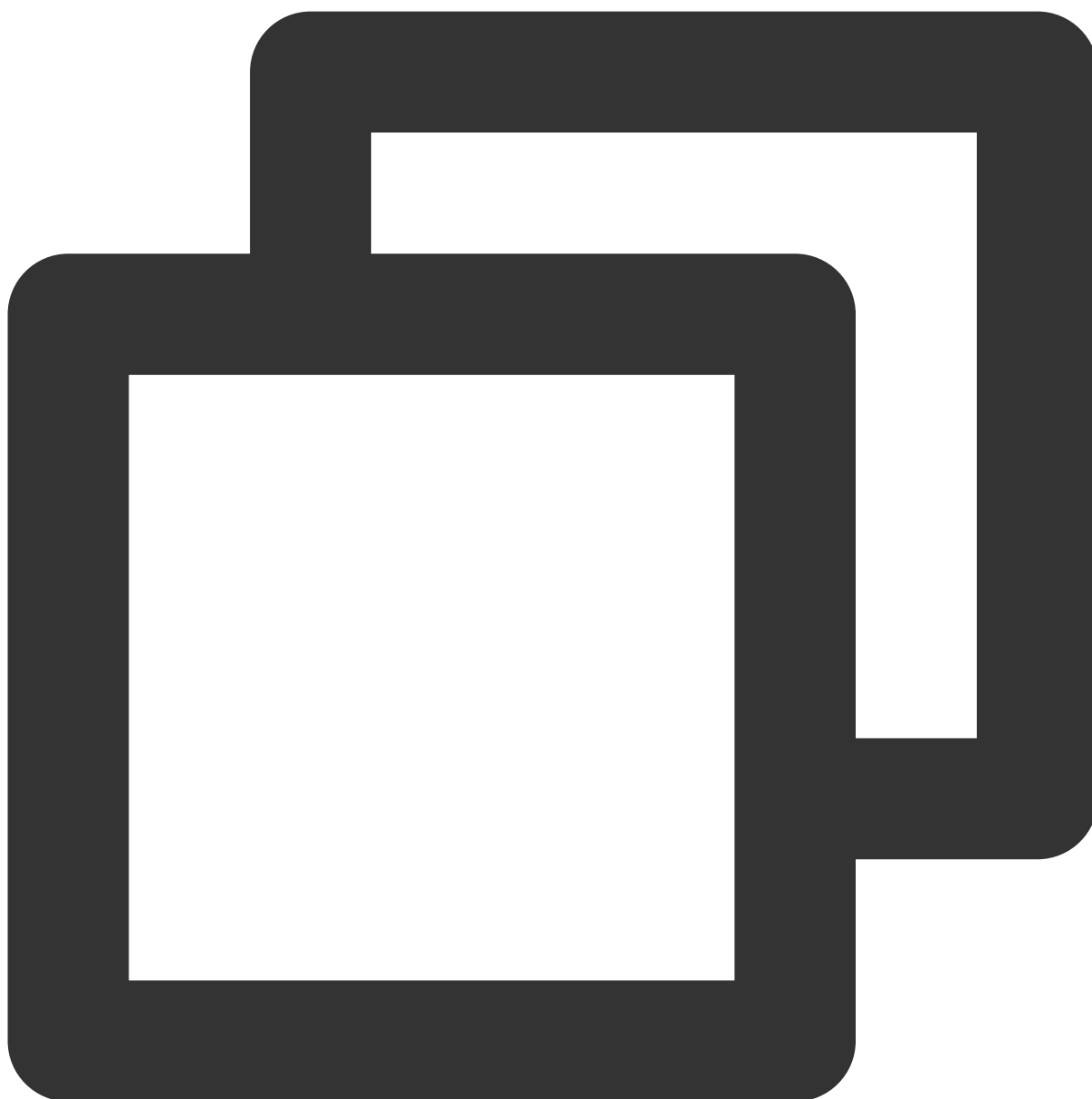
最近更新时间：2024-07-24 09:58:38

打开小程序

打开小程序时，会先判断本地是否有缓存的小程序，如果没有，则会自动从远程服务器上下载小程序，然后打开。如果有缓存的小程序，则会先打开本地小程序，同时后台校验服务器端是否有新版本。

说明：

如果有新版本，则下载新版小程序，下次打开时，就会使用新版小程序。



```
// 通过小程序id打开小程序
// Open the mini program through the mini program id
// @param appID 小程序ID - Mini program ID
// @param verType 指定打开的小程序类型 - The version type of mini program to open
// @param scene 场景值 - scene value
// @param firstPage 打开页面 - open the page
// @param paramsStr 带入参数 - brings in parameters
// @param parentVC 从哪个vc呼起 - Which vc to call from
// @param completion 错误回调 - error callback
- (void)startUpMiniAppWithAppID:(NSString *)appID
                        verType:(TMAVersionType)verType
```

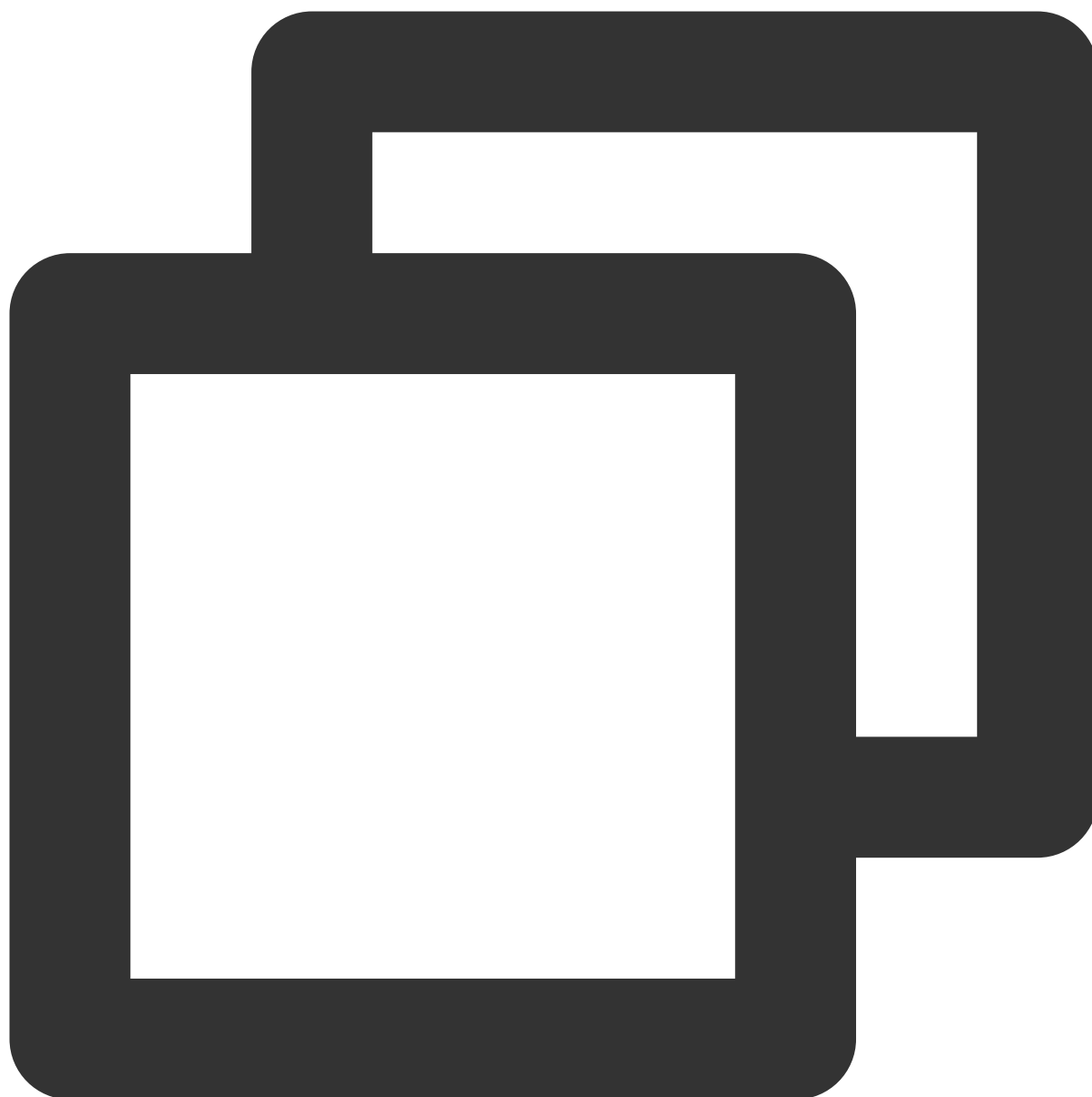
```
scene: (TMAEntryScene) scene
firstPage: (NSString * _Nullable) firstPage
paramsStr: (NSString * _Nullable) paramsStr
parentVC: (UIViewController *) parentVC
completion: (void (^) (NSError * _Nullable)) completion;
```

options 支持的参数列表：

名称	是否必填	类型	作用
appId	YES	NSString	打开指定小程序的小程序 id
verType	YES	TMAVersionType	打开小程序的类型
scene	YES	TMAEntryScene	打开小程序使用的场景值
firstPage	NO	NSString	打开页面
paramsStr	NO	NSString	打开传递参数
parentVC	YES	UIViewController	从哪个 VC 呼起
completion	YES	block	错误回调

根据小程序ID（appId）打开小程序

打开正式版本（通过审核并已经发布上线）的小程序。参考如下代码：

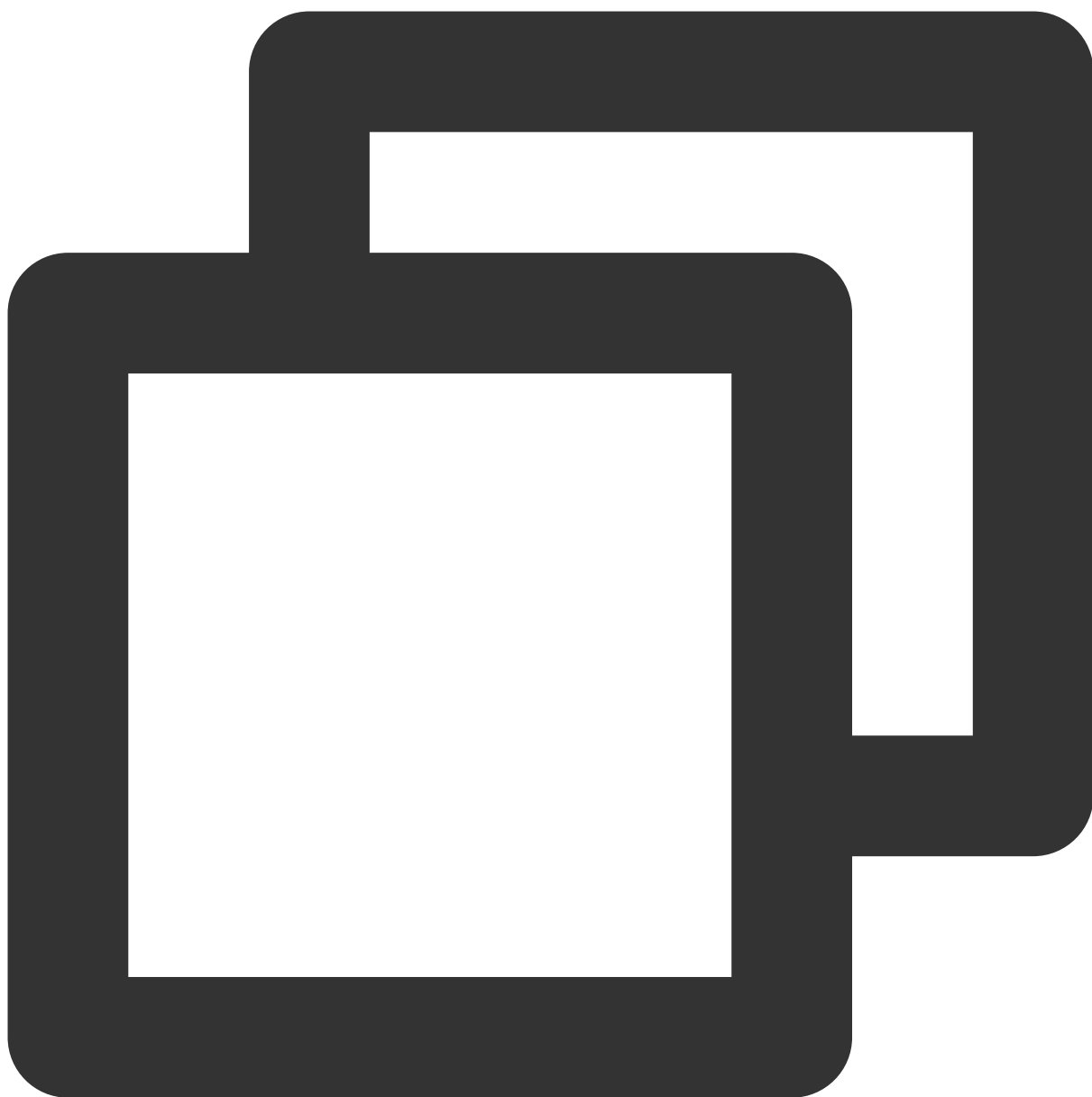


```
[[TMFMiniAppSDKManager sharedInstance] startUpMiniAppWithAppID:appId parentVC:self  
  NSLog(@"open applet error:%@",error);  
}];
```

说明：

appId 字段为小程序的 ID，需要通过小程序开发者或者小程序搜索接口获取。

打开预览版或开发版小程序，示例代码：



```
[[TMFMiniAppSDKManager sharedInstance] startUpMiniAppWithAppID:appId verType:verType  
NSLog(@"open applet error:%@",error);  
}];
```

注意：

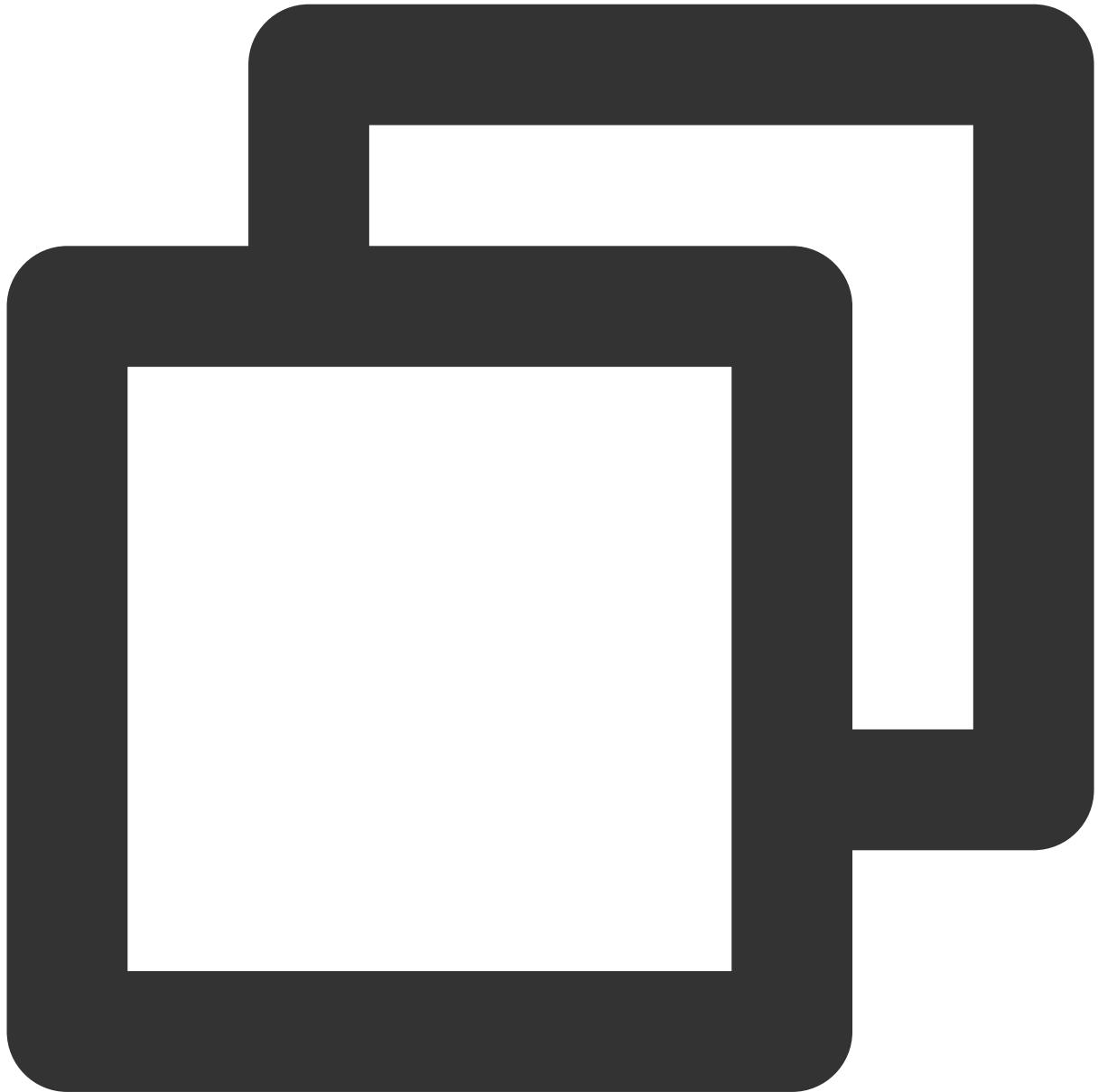
appVerType 应当和小程序的版本匹配；一般可以通过接口（getRecentList）返回的 TMFAppletSearchInfo 对象实例获取 appVerType 的值。

预览版小程序的 appVerType 参数值应当为 TMAVersionPreview。

开发版小程序的 appVerType 参数值应当为 TMAVersionDevelop。

调用扫码打开小程序

小程序 SDK 提供根据扫码打开小程序的接口；在使用小程序 SDK 提供的扫码能力前，需要提前接入扩展库 TCMPPExtScanCode。

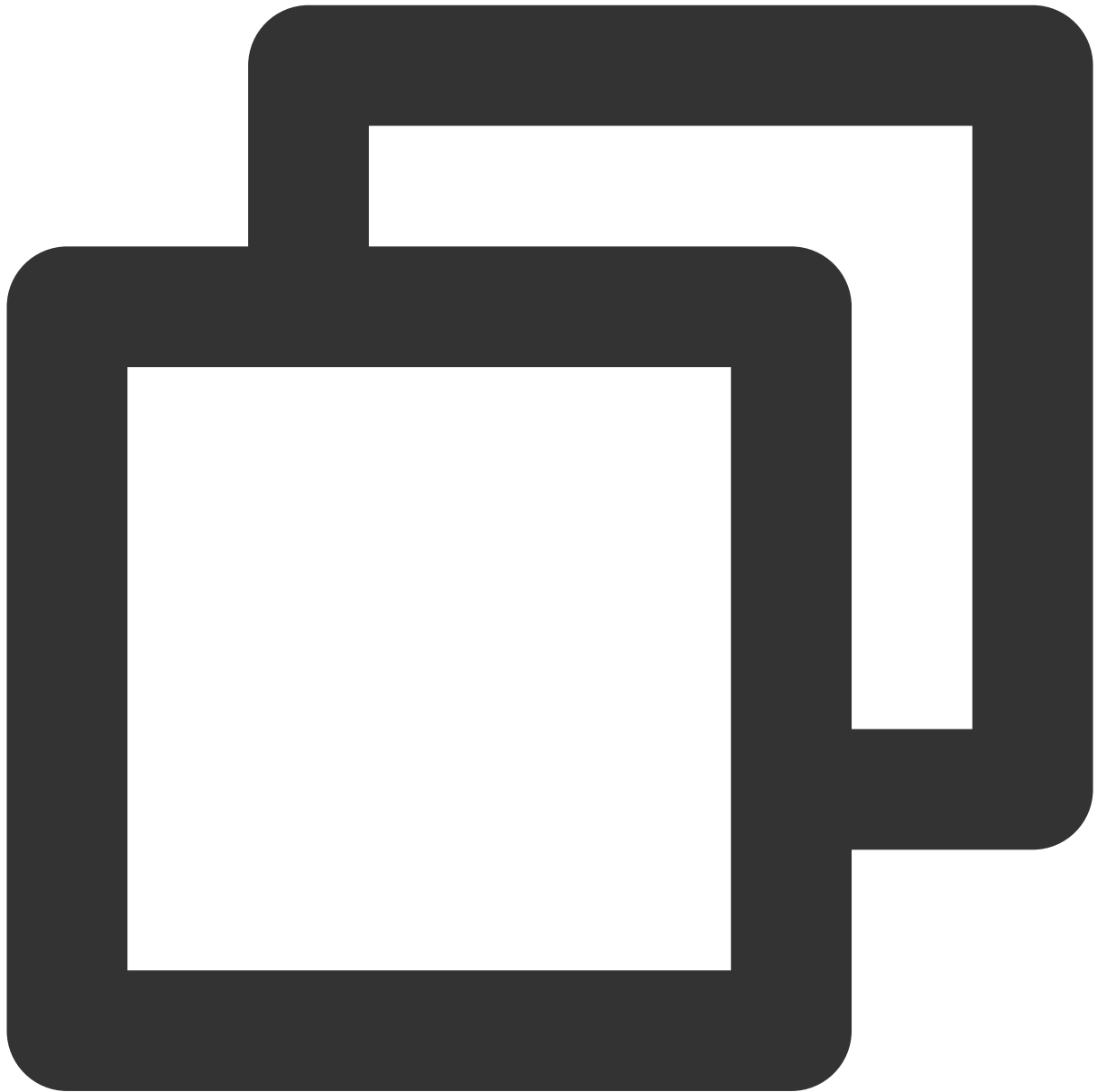


```
// 通过扫码拉起小程序 - Quickly open the mini program through QRCode
// @param parentVC 从哪个vc呼起 - Which vc to call from
// @param completion 错误回调 - error callback
- (void)startUpMiniAppWithQRCodeWithParentVC:(UIViewController *)parentVC
                                completion:(void (^)(NSError * _Nullable))completion;
```


根据二维码内容打开小程序

小程序 SDK 提供根据平台二维码内容打开小程序的接口；在使用小程序 SDK 提供的扫码能力前，需要提前接入扫一扫的扩展能力，扫码能力的接入可以参考：扫码能力支持。

接入扫码能力后，可以通过如下的方法启动二维码扫描并打开小程序：



```
/// 通过二维码唤起小程序
/// @param qrData 二维码内容
/// @param parentVC 从哪个vc唤起
/// @param completion 错误回调
- (void)startUpMiniAppWithQrData:(NSString *)qrData
```

```
parentVC:(UIViewController *)parentVC  
completion:(void (^)(NSError * _Nullable error))completion;
```

关闭小程序

最近更新时间：2024-07-03 18:01:14

关闭小程序

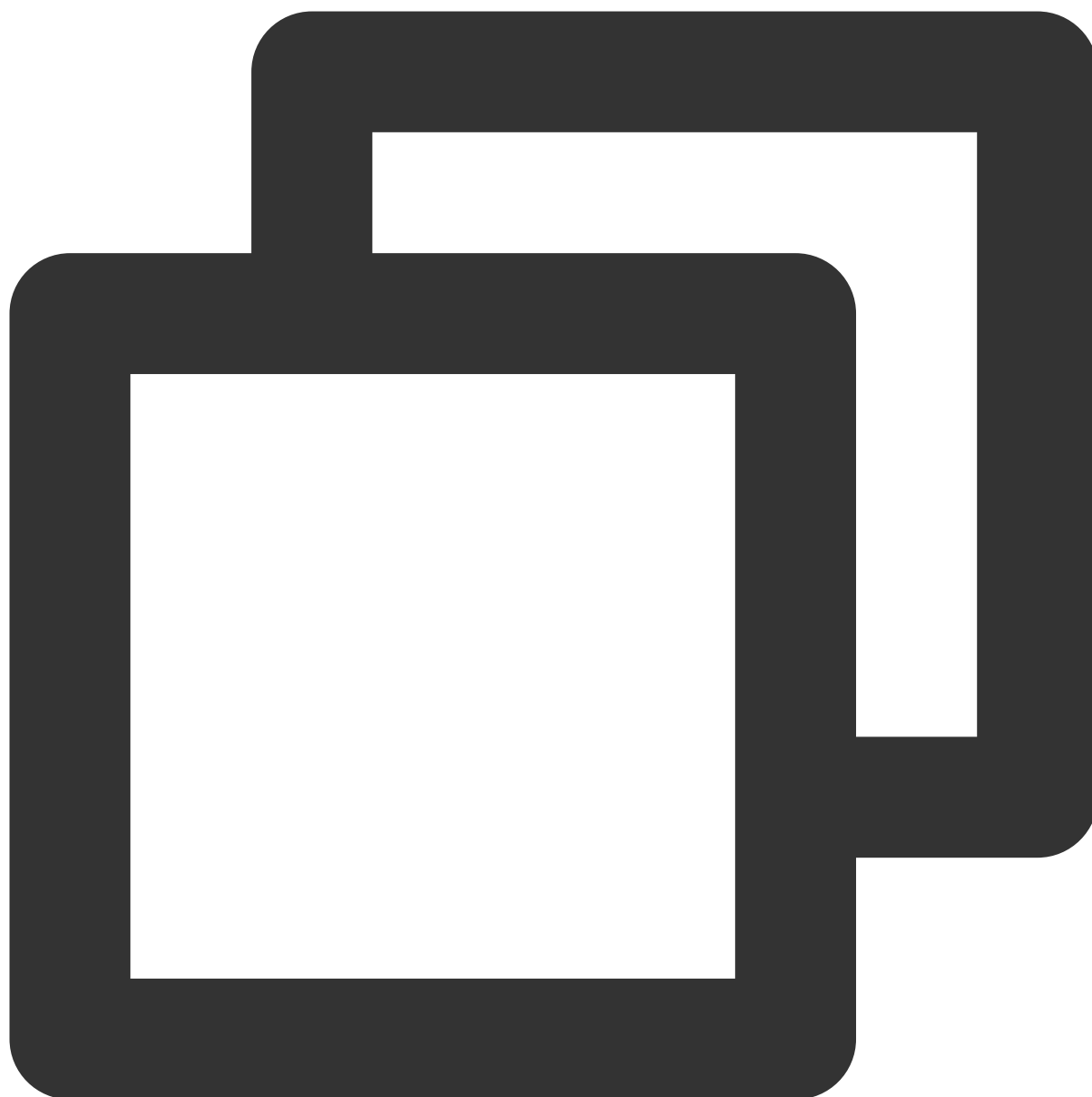
说明：

关闭小程序指仅关闭小程序页面，小程序退到后台，即仍然存在于内存中，下次打开时，可以迅速从内存中唤起，耗时较少。

注意：

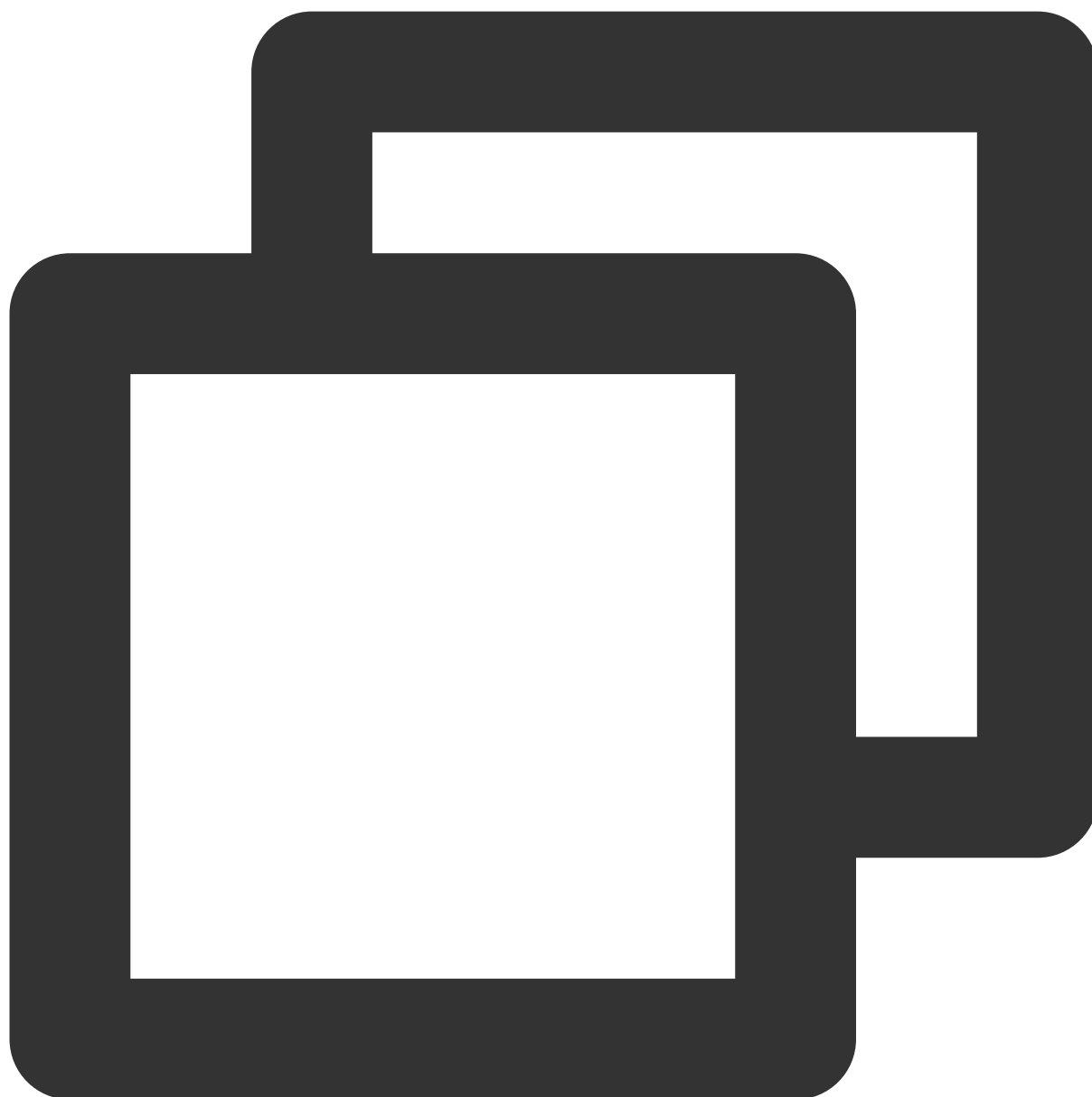
1. 小程序关闭后默认保活10分钟；
2. 同时最多保活3个小程序。

1. 关闭指定小程序



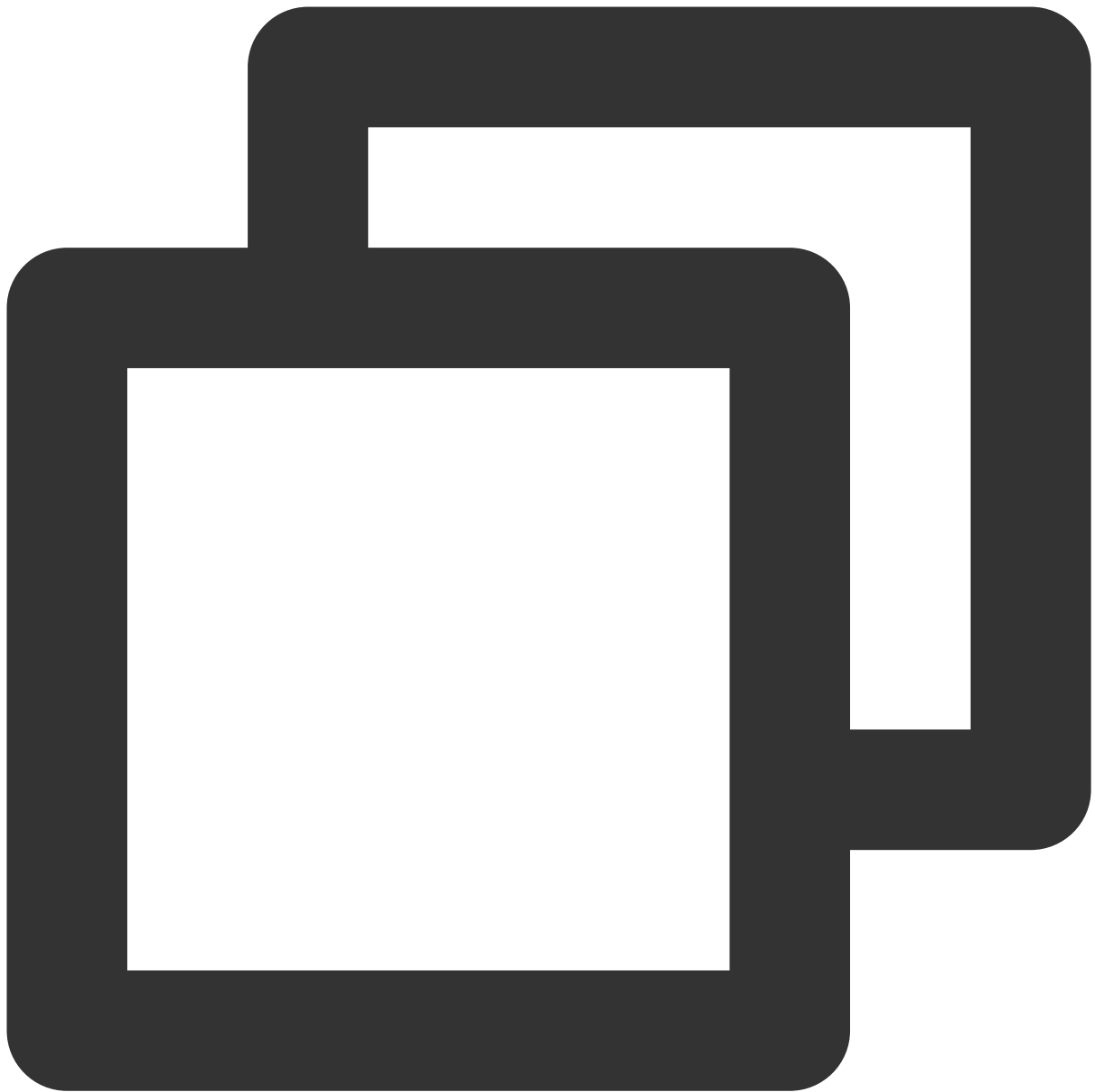
```
/// 关闭指定的小程序对象  
/// @param appID 要关闭的 appID  
- (void)closeMiniAppWithAppID:(NSString *)appID;
```

2. 关闭当前小程序



```
/// 关闭当前正在运行的小程序对象  
- (void)closeCurrentApplet;
```

3. 关闭所有打开的小程序



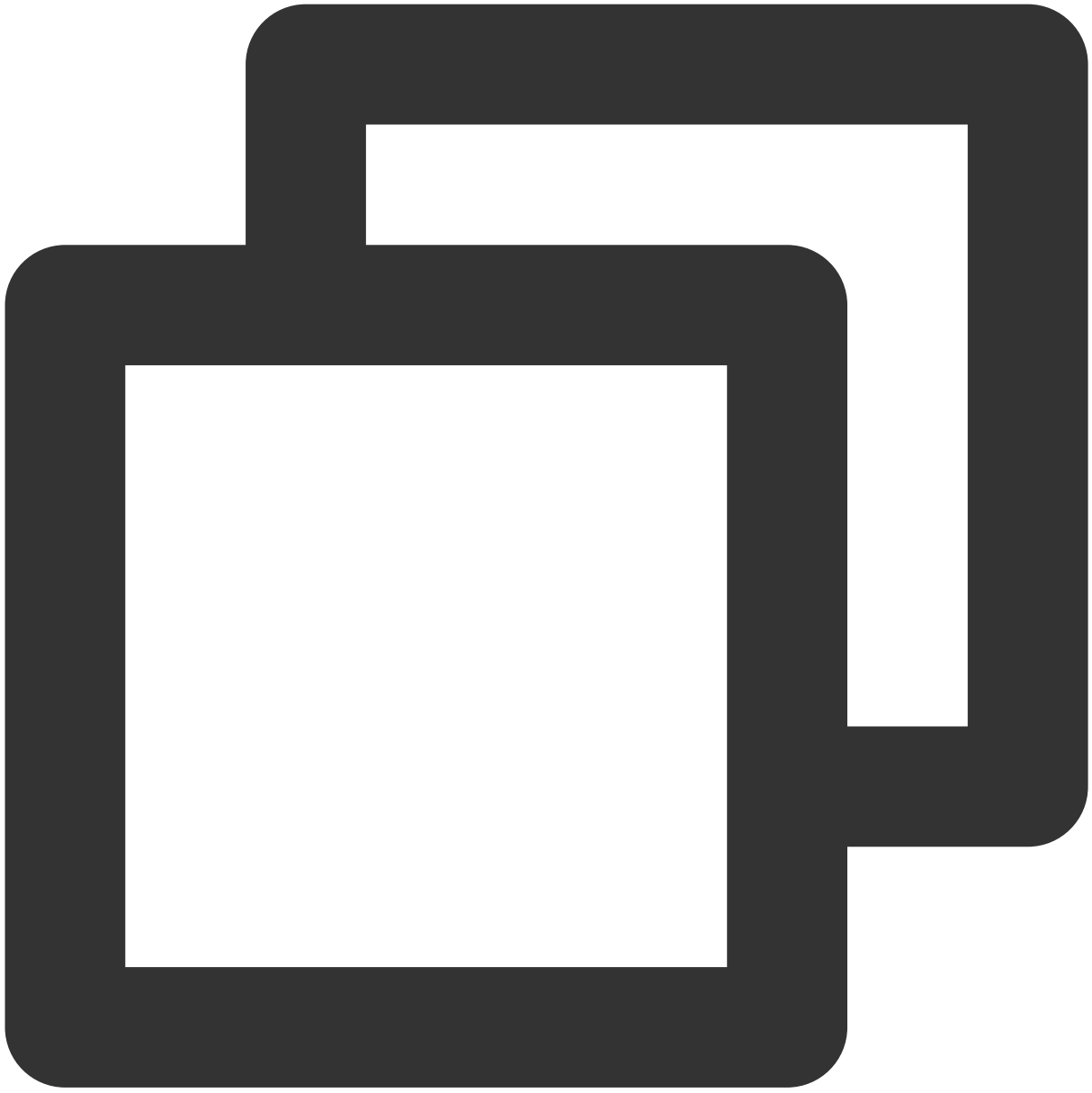
```
/// 关闭所有内存中正在运行的小程序  
- (void)closeAllApplications;
```

终止小程序

说明：

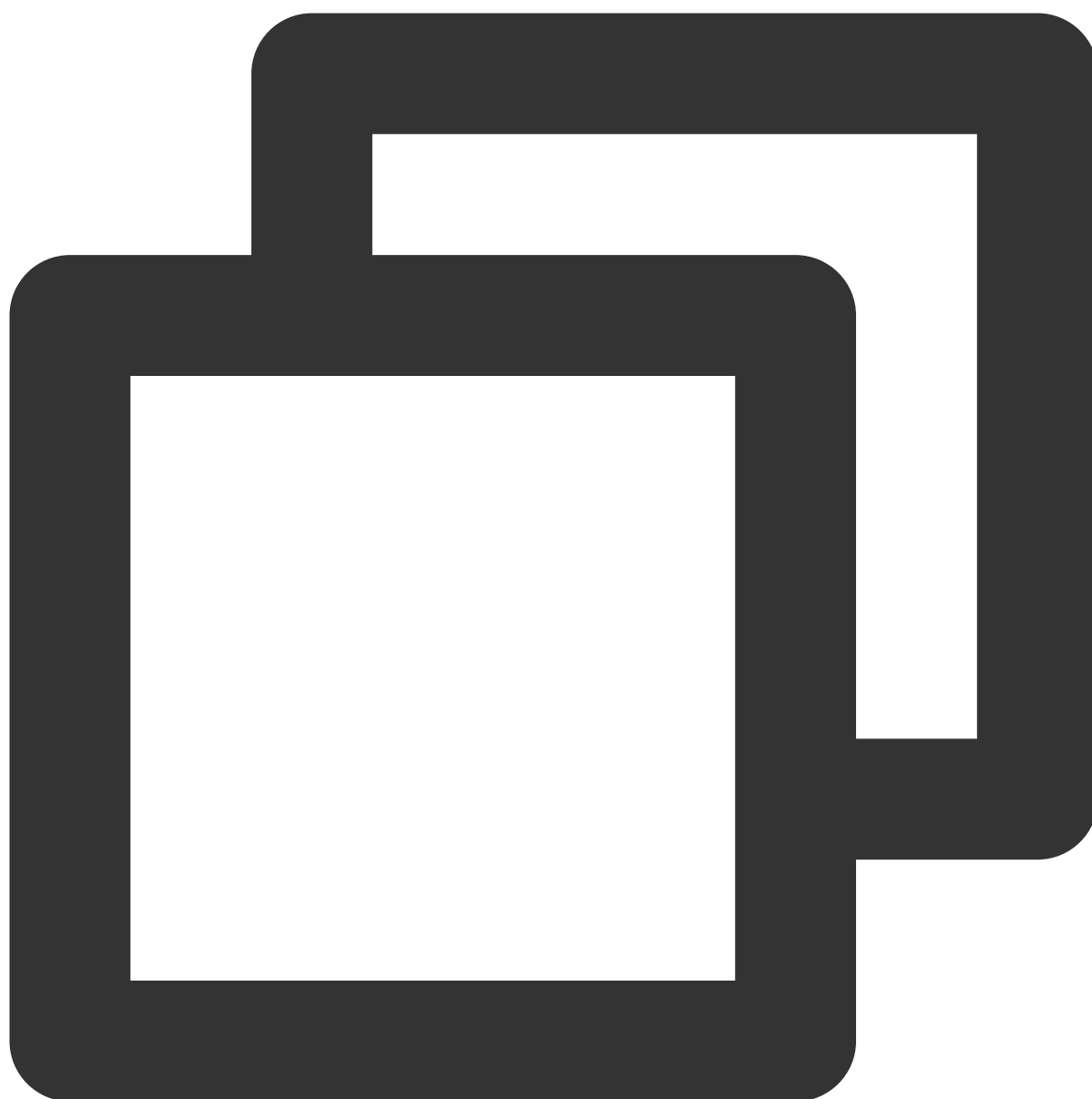
终止小程序指在内存中清除小程序，下次打开时，需要重新将小程序加载到内存，耗时较多。

1. 终止指定小程序



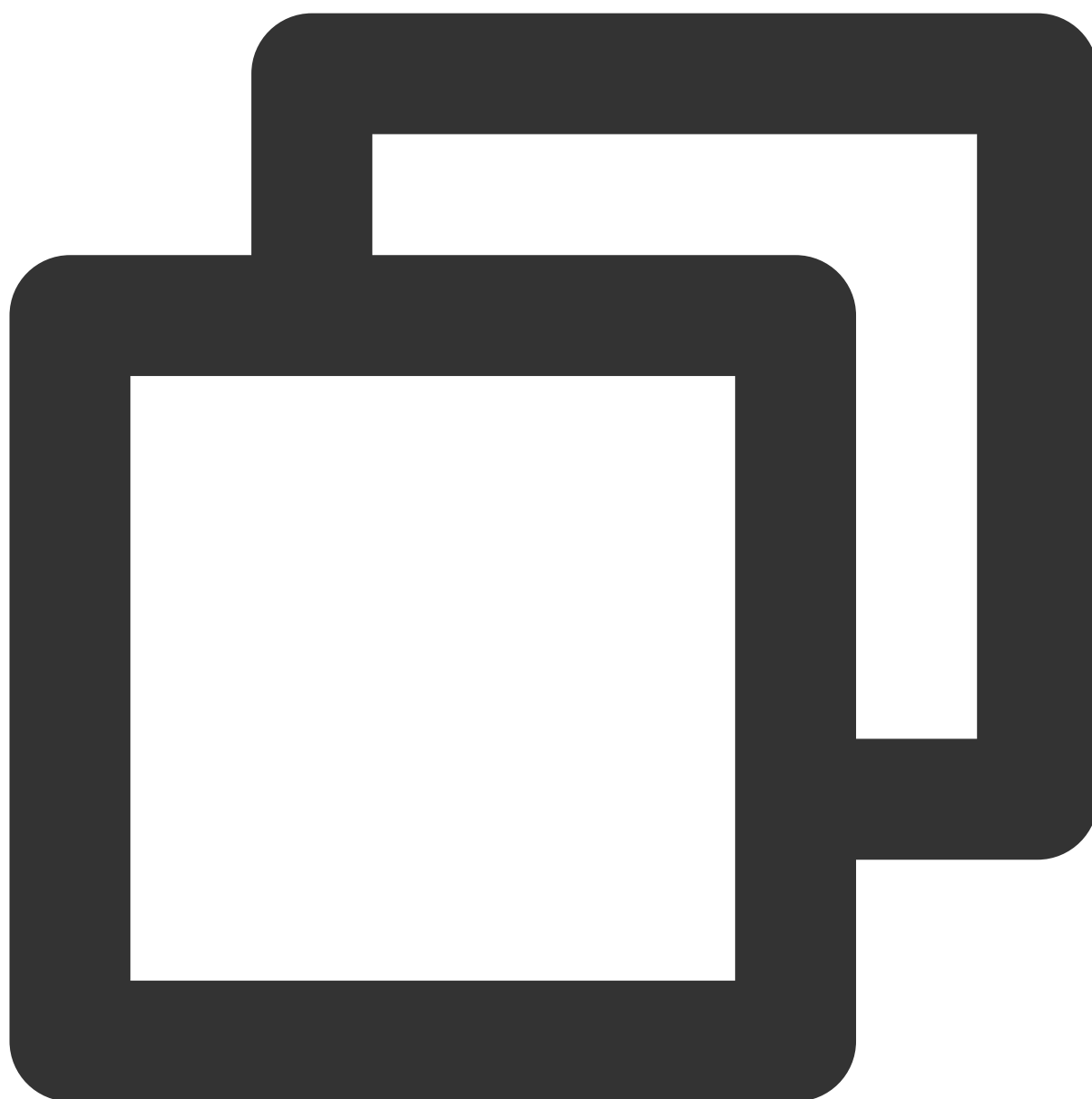
```
/// 终止指定的小程序对象  
/// @param appID 要终止的 appID，默认删除所有版本  
- (void)terminateMiniAppWithAppID:(NSString *)appID;
```

2. 终止当前小程序



```
/// 终止当前正在运行的小程序对象  
- (void)terminateCurrentApplet;
```

3. 终止所有小程序



```
/// 终止栈上与保活中的小程序  
- (void)terminateAllApplications;
```

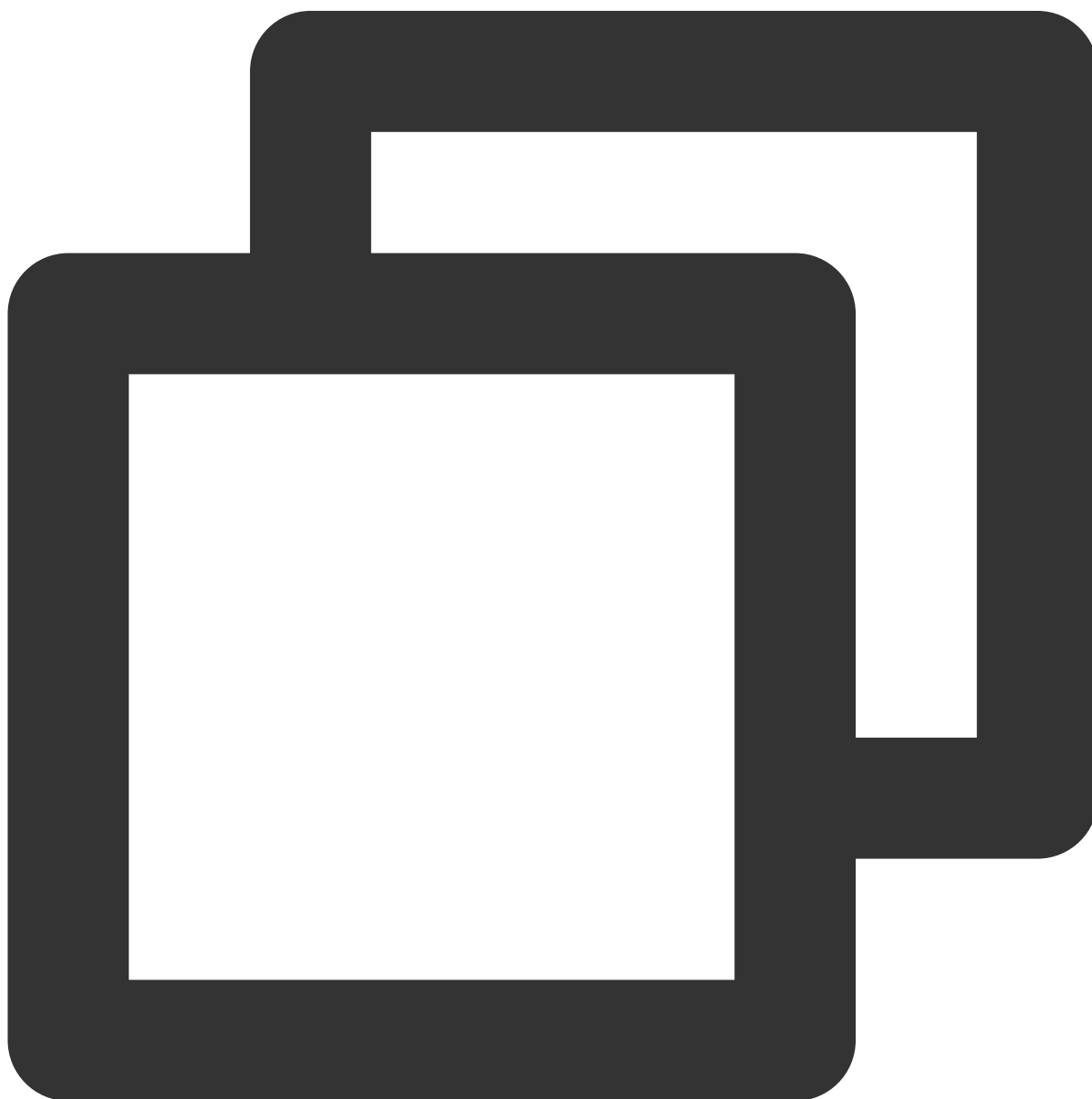
删除小程序

最近更新时间：2024-07-03 18:01:49

说明：

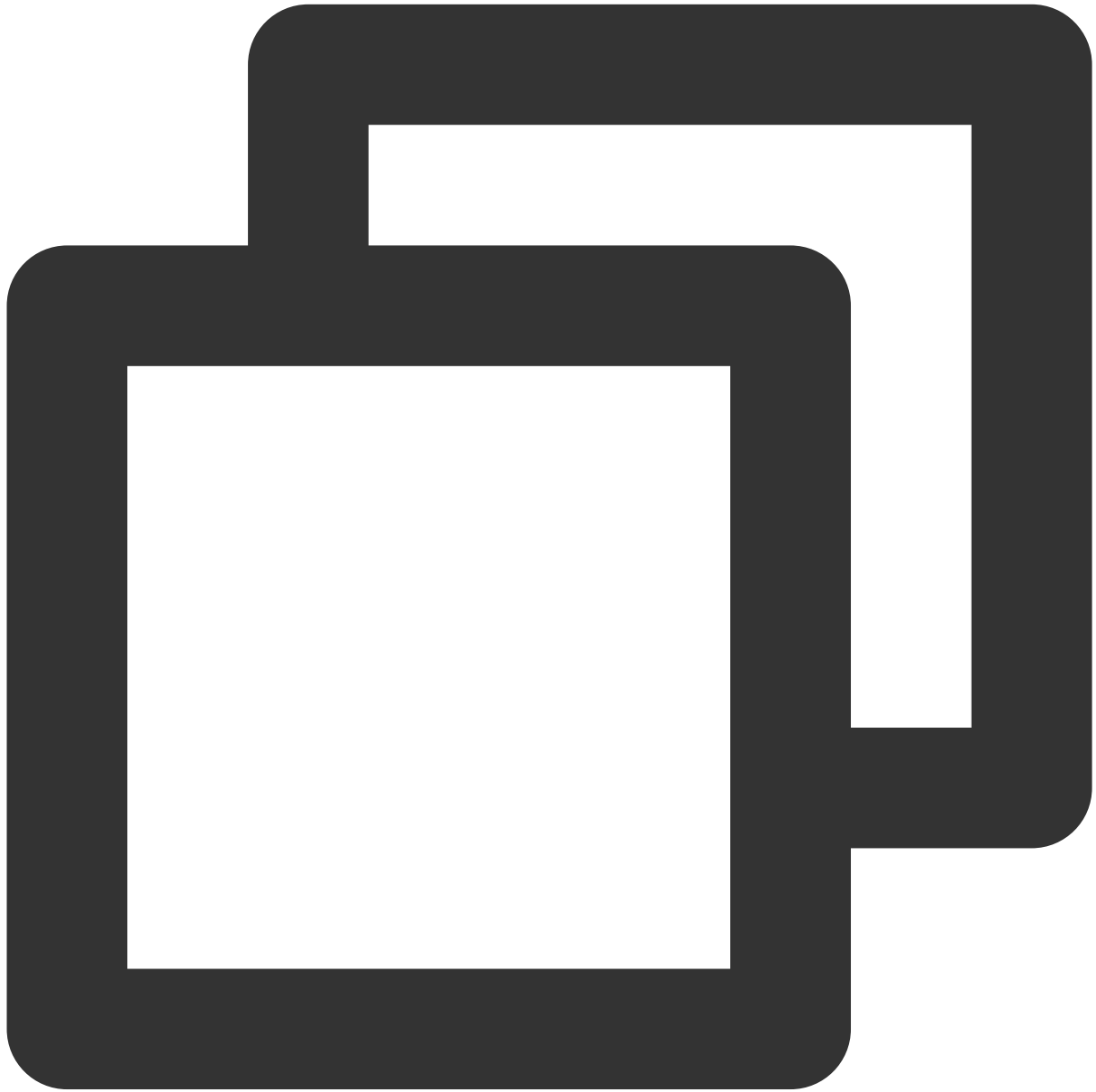
由于小程序的运行，会将小程序包和小程序信息缓存在本地，以便下次更快速的打开。如果想要将小程序的所有信息都删除，可以调用以下 API 删除某个小程序或者删除所有小程序。

删除所有小程序缓存



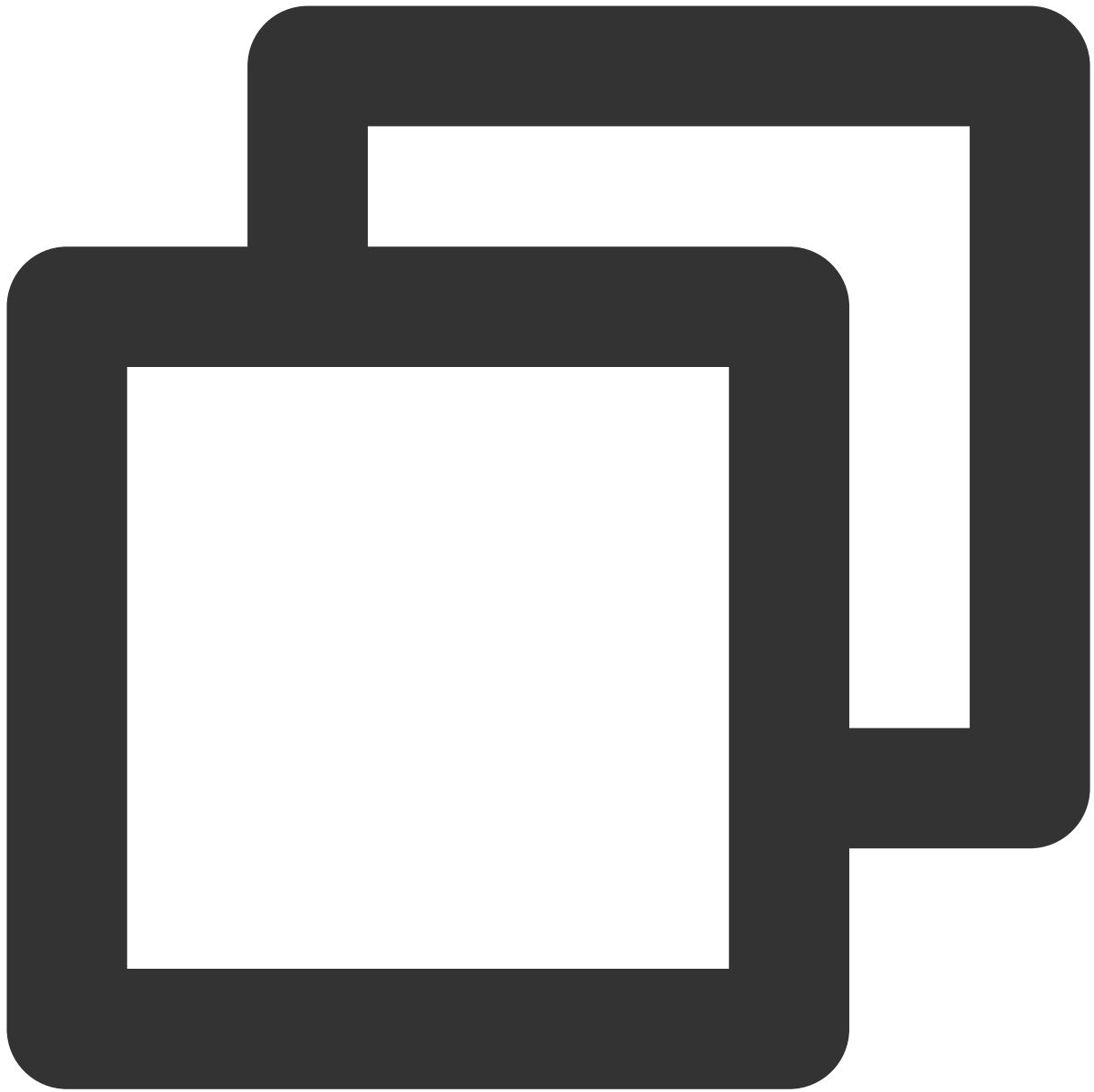
```
/// 移除小程序所有相关缓存 包含资源包、基础库、小程序/小游戏沙箱数据  
- (void)clearMiniAppCache;
```

删除指定小程序缓存



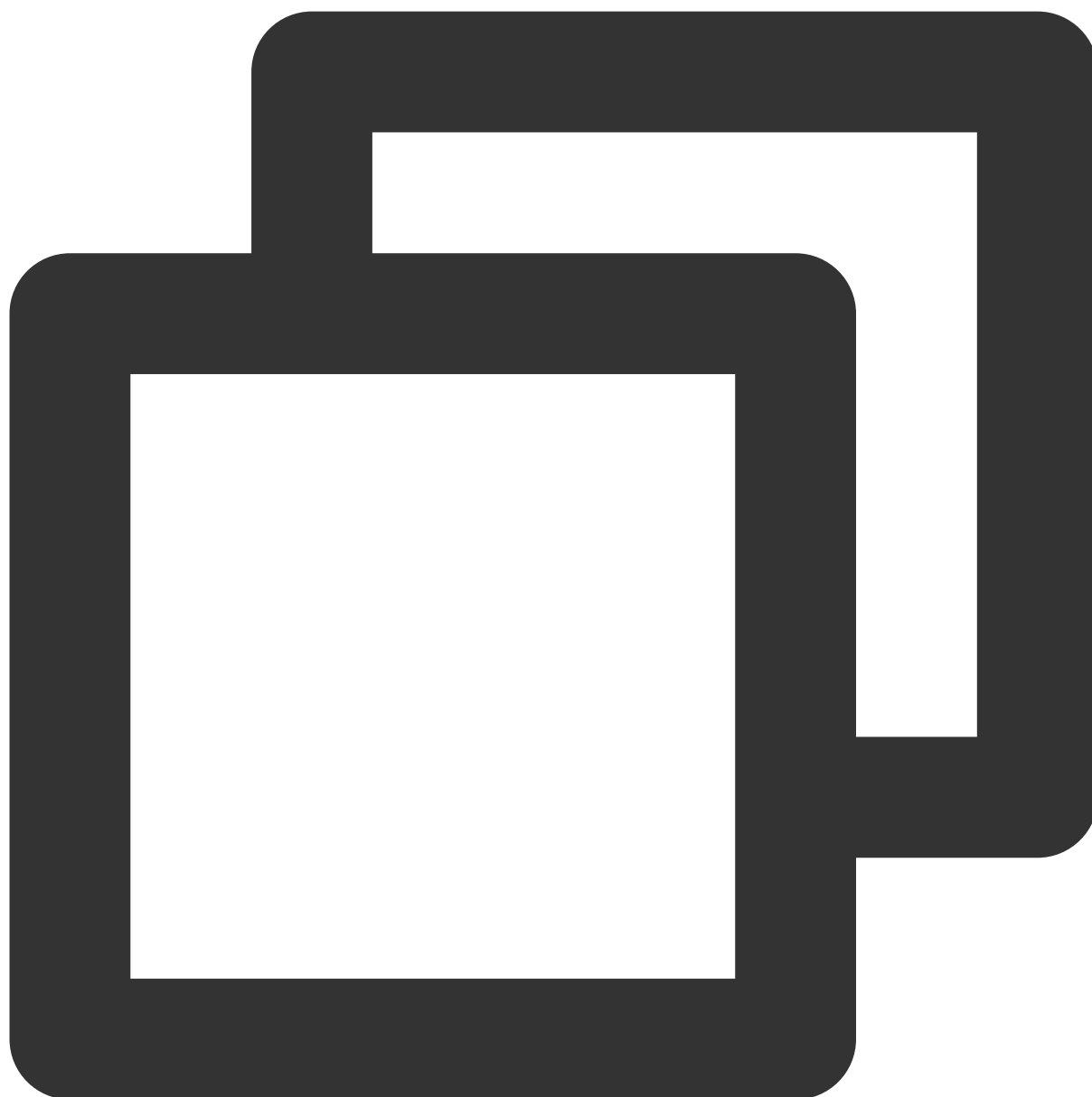
```
/// 删除本地缓存小程序  
/// @param appID 要删除的 appID, 默认删除所有版本  
- (void)clearCacheWithAppID:(NSString *)appID;
```

删除指定版本和类型的小程序



```
// 删除本地缓存小程序
// Delete local cache mini programs
// @param appID 要删除的 appID, 默认删除所有版本 - The appID to be deleted, all version
// @param verType 要删除的本地缓存小程序类型 - The version type of mini program to delete
- (void)clearCacheWithAppID:(NSString *)appID verType:(TMAVersionType)verType;
```

示例代码：

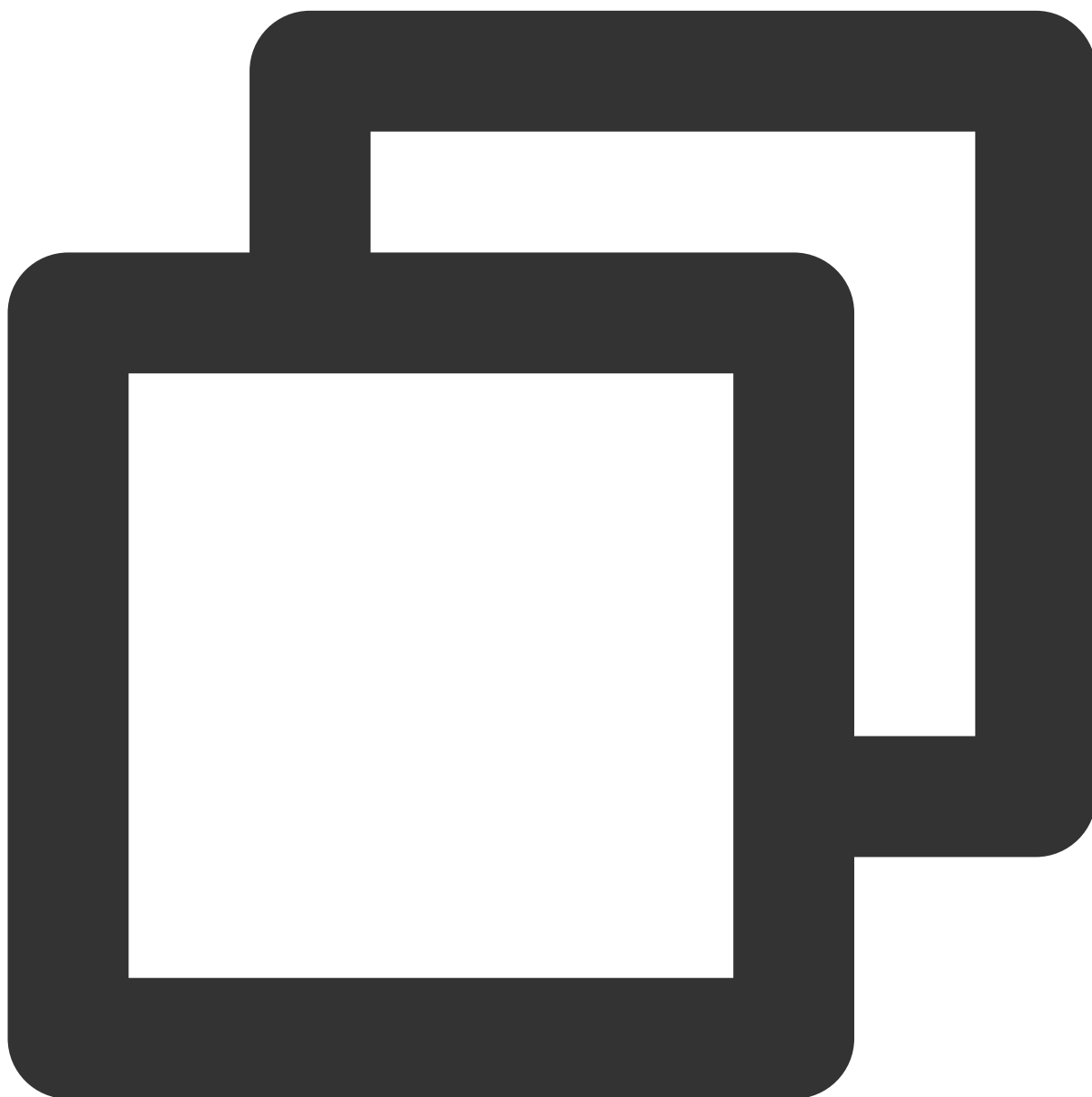


```
[[TMFMiniAppSDKManager sharedInstance] clearCacheWithAppID:appId verType:verType];
```

获取小程序最近访问列表及信息

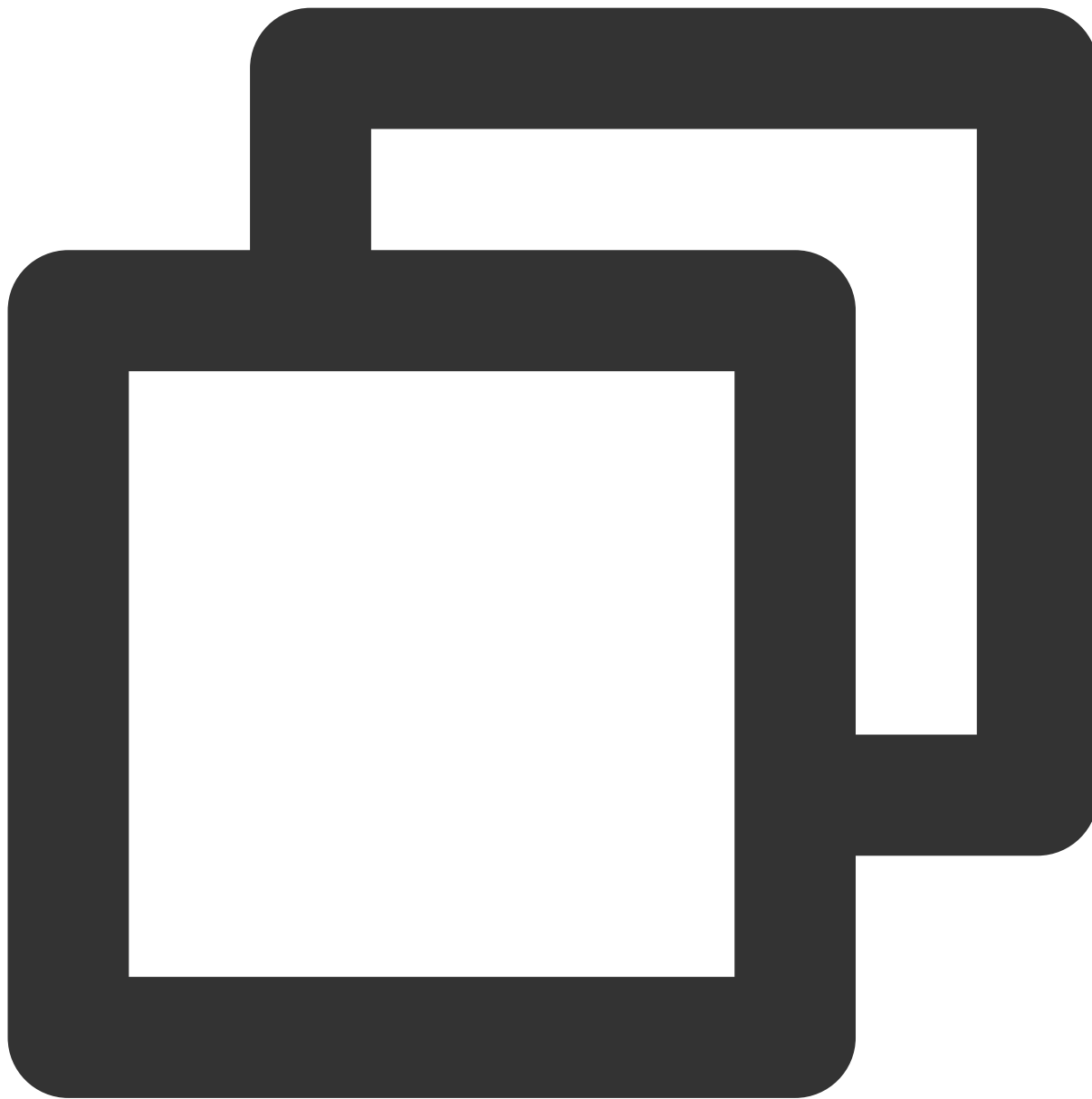
最近更新时间：2024-07-03 18:02:31

获取最近打开过的小程序信息



```
//获取最近打开的所有小程序信息
///@return 小程序数组<TMFMiniAppInfo>
- (NSArray *)loadAppletsFromCache;
```

获取当前正在运行的小程序信息

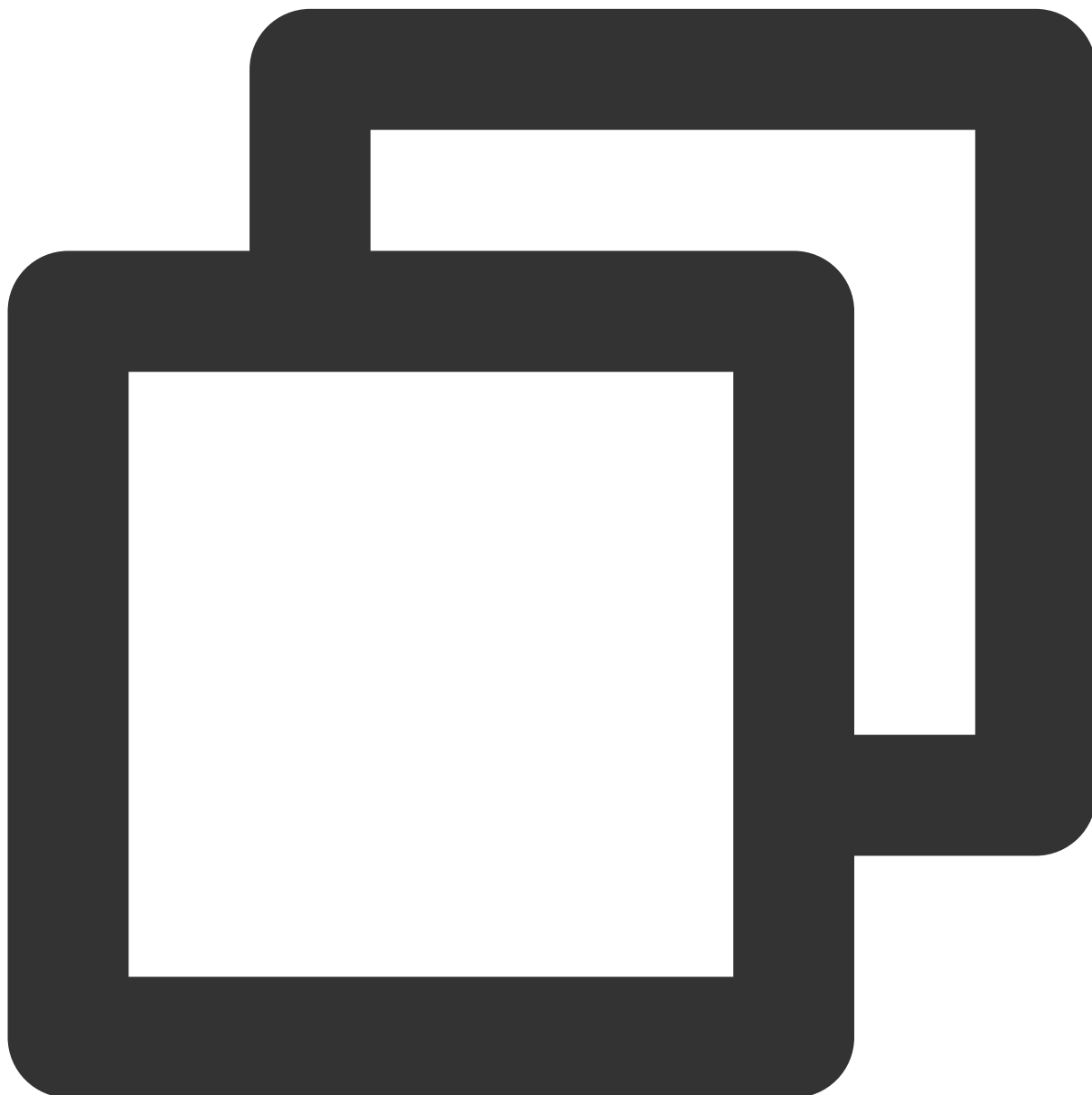


```
/// 获取当前正在运行的小程序对象  
/// @return TMFAppletInfo 小程序信息  
- (TMFMiniAppInfo *)currentApplet;
```

预下载小程序

最近更新时间：2024-07-03 18:02:40

由于小程序在打开时会同步检测和更新版本，小程序在打开过程中可能存在等待的过程；为了降低小程序打开的等待时间，优化用户体验，提供了如下的接口用于预下载小程序包。



```
///预准备小程序信息
///@param appId 要准备小程序信息
///@param isDownload 是否直接下载
```



```
///@param complete 批量更新小程序回调  
- (void)preloadMiniApps:(NSArray *)appIds isDownload:(BOOL)isDownload complete:(voi
```

小程序文件管理

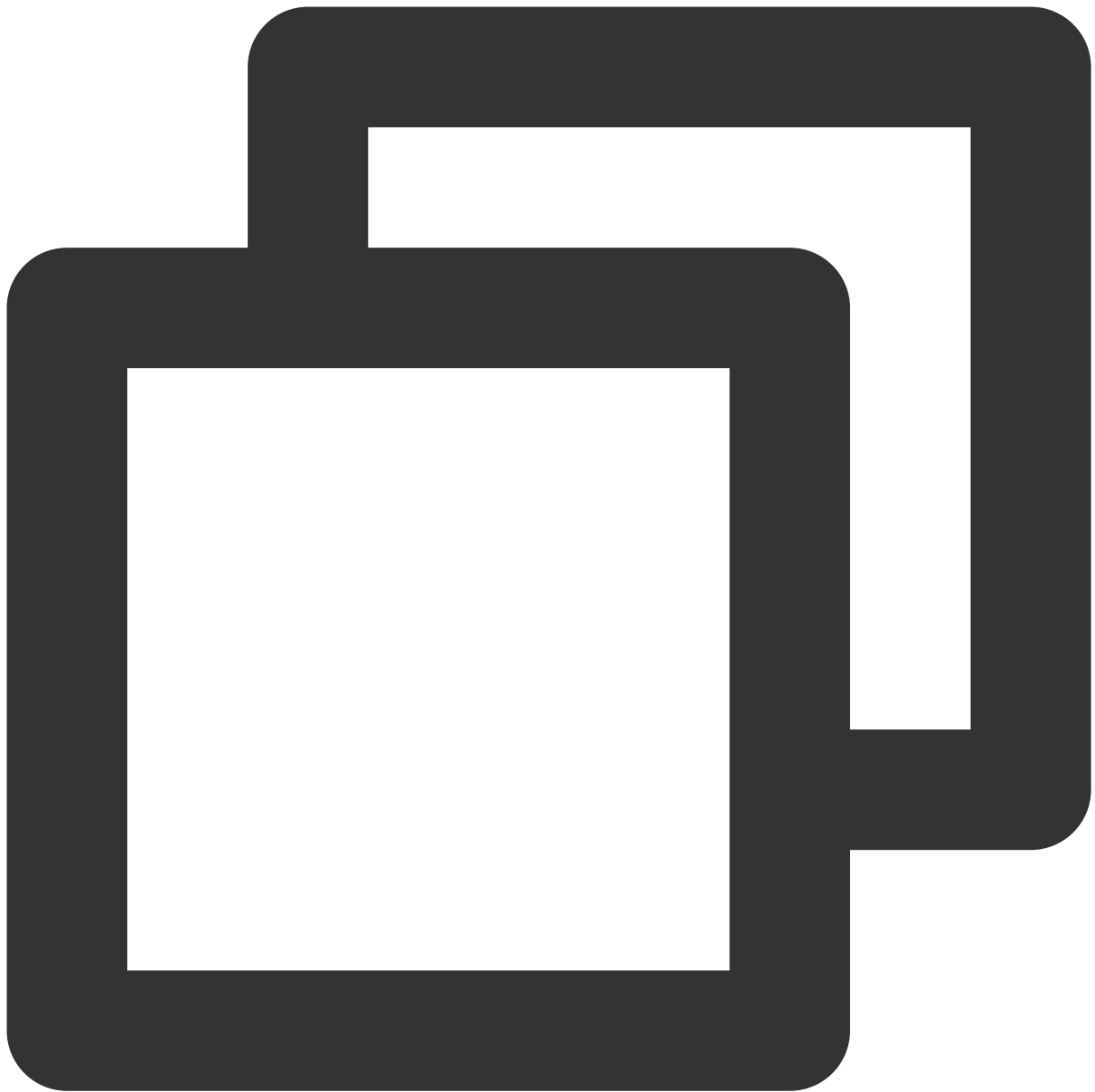
最近更新时间：2024-07-03 18:02:58

wxfile 路径转换为绝对路径

在有些场景下，我们拿到小程序的文件路径，无法获取到文件数据，这时可以用该方法转换为绝对路径。

例如，在使用小程序的转发功能时，返回的图片路径是小程序文件路径，可以用该方法转换为绝对路径，然后获取到图片数据再去调起第三方分享。再例如，自定义 API 里，可以传递小程序文件路径的参数，宿主 App 用该方法转换为绝对路径后，即可拿到文件数据。

示例代码：

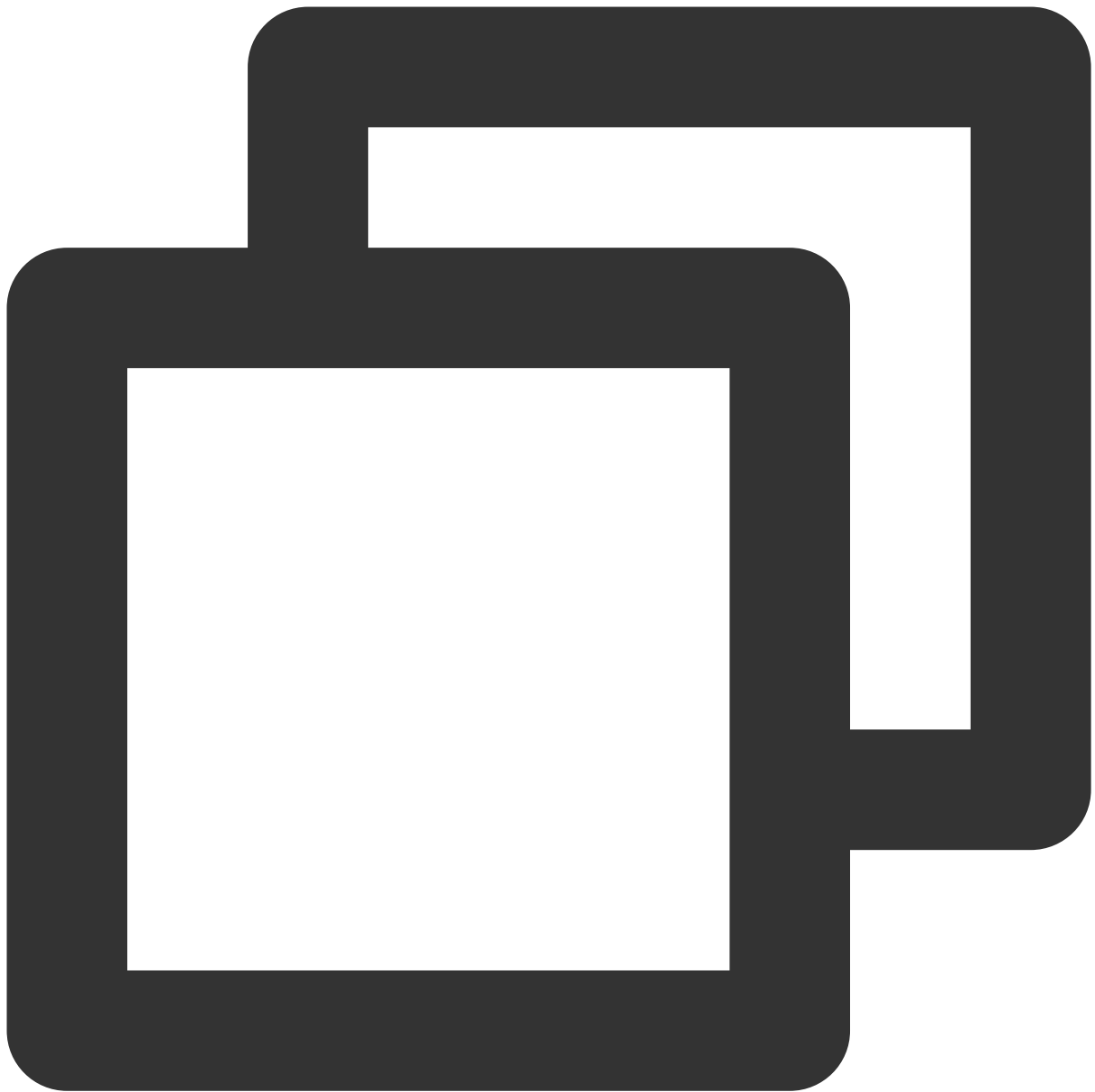


```
TMAFileManager *fm = [[TMFMiniAppSDKManager sharedInstance] getFileManagerWithAppID];
NSString *filePath = [fm translateWxfilePathToAbsolutePath:wxPath];
```

绝对路径转换成 wxfile 路径

SDK 支持原生在小程序的缓存目录中创建的文件路径，转换成小程序使用的文件路径返回，供小程序内部使用。

示例代码：

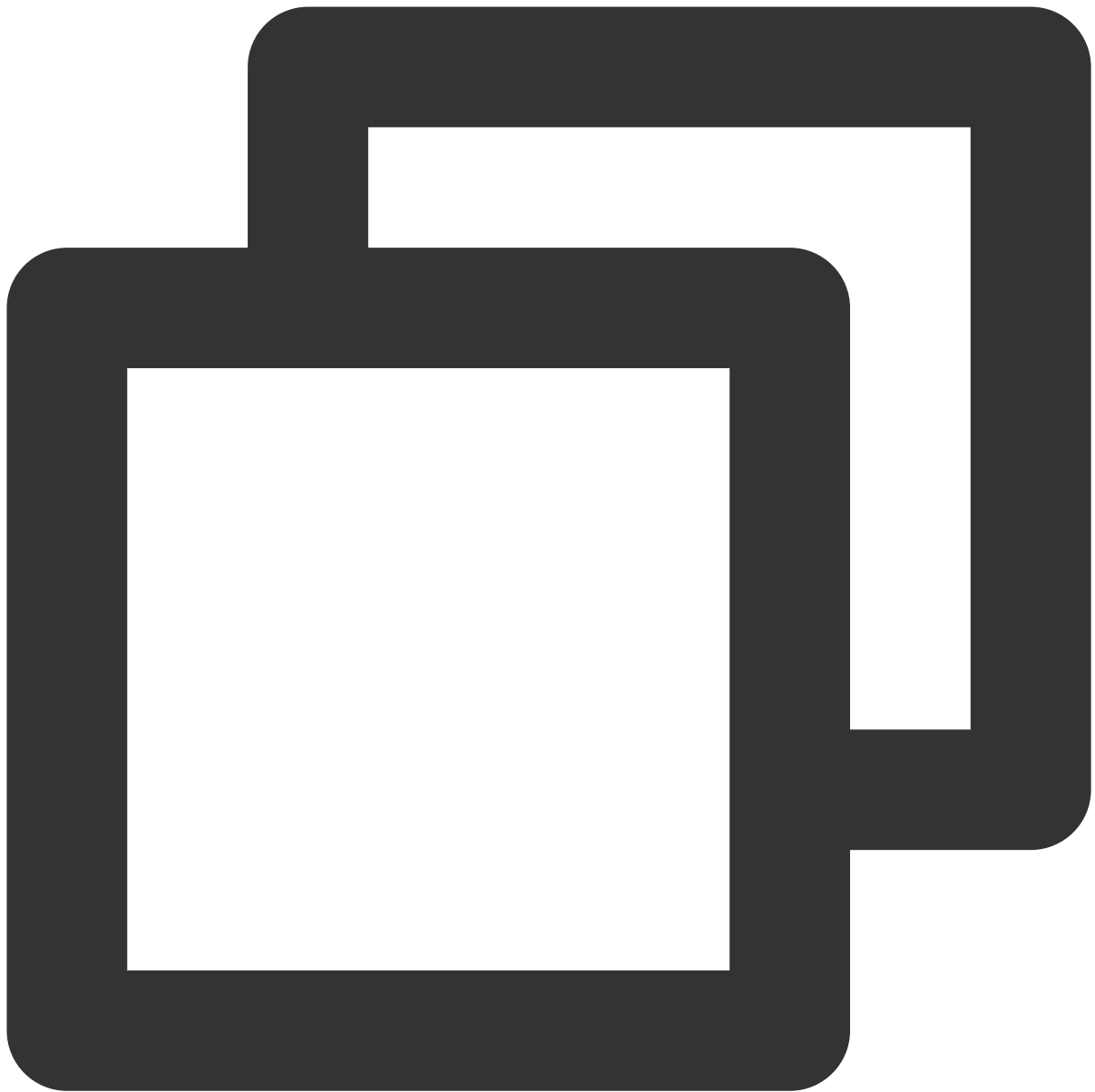


```
TMAFileManager *fm = [[TMFMiniAppSDKManager sharedInstance] getFileManagerWithAppID  
NSString *tmpPath = [fm createMediaTmpPathWithFileName:@"a.pdf" type:TMATmpPathType  
  
NSString *wxPath = [fm translateAnyPathToWxfilePath:tmpPath];
```

在小程序临时目录创建文件

SDK 支持原生在小程序的缓存目录创建文件，然后返回文件的本地路径，以供宿主使用。

示例代码：



```
TMAFileManager *fm = [[TMFMiniAppSDKManager sharedInstance] getFileManagerWithAppID  
NSString *tmpPath = [fm createMediaTmpPathWithFileName:@"a.pdf" type:TMATmpPathType
```

预置离线小程序

最近更新时间：2024-07-03 18:03:07

离线小程序是内置在宿主 App 内的小程序，需要将小程序包从控制台下载下来，导入到宿主 App 工程中，跟随 App 一起打包。在 App 使用过程中，用户打开某个内置的小程序，无需从后端拉取下载，即使在没有网络连接的情况下也能打开并运行的小程序。

预置流程

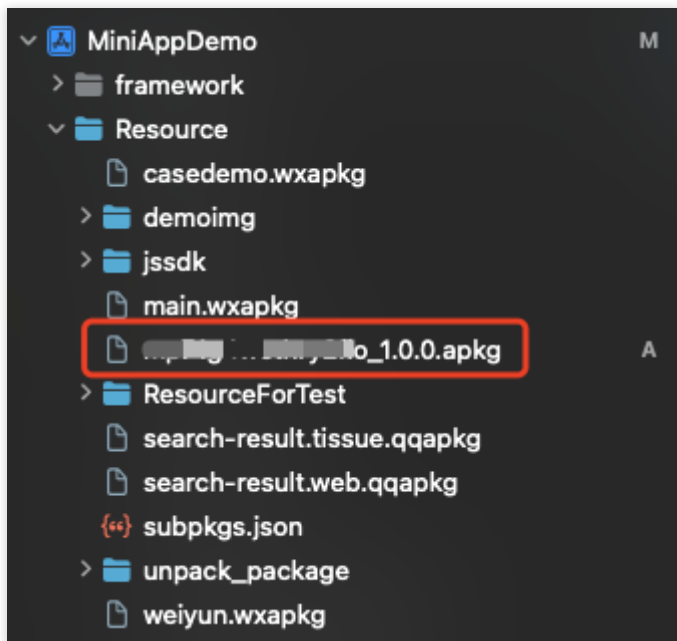
1. 从控制台下载需要预置的小程序。



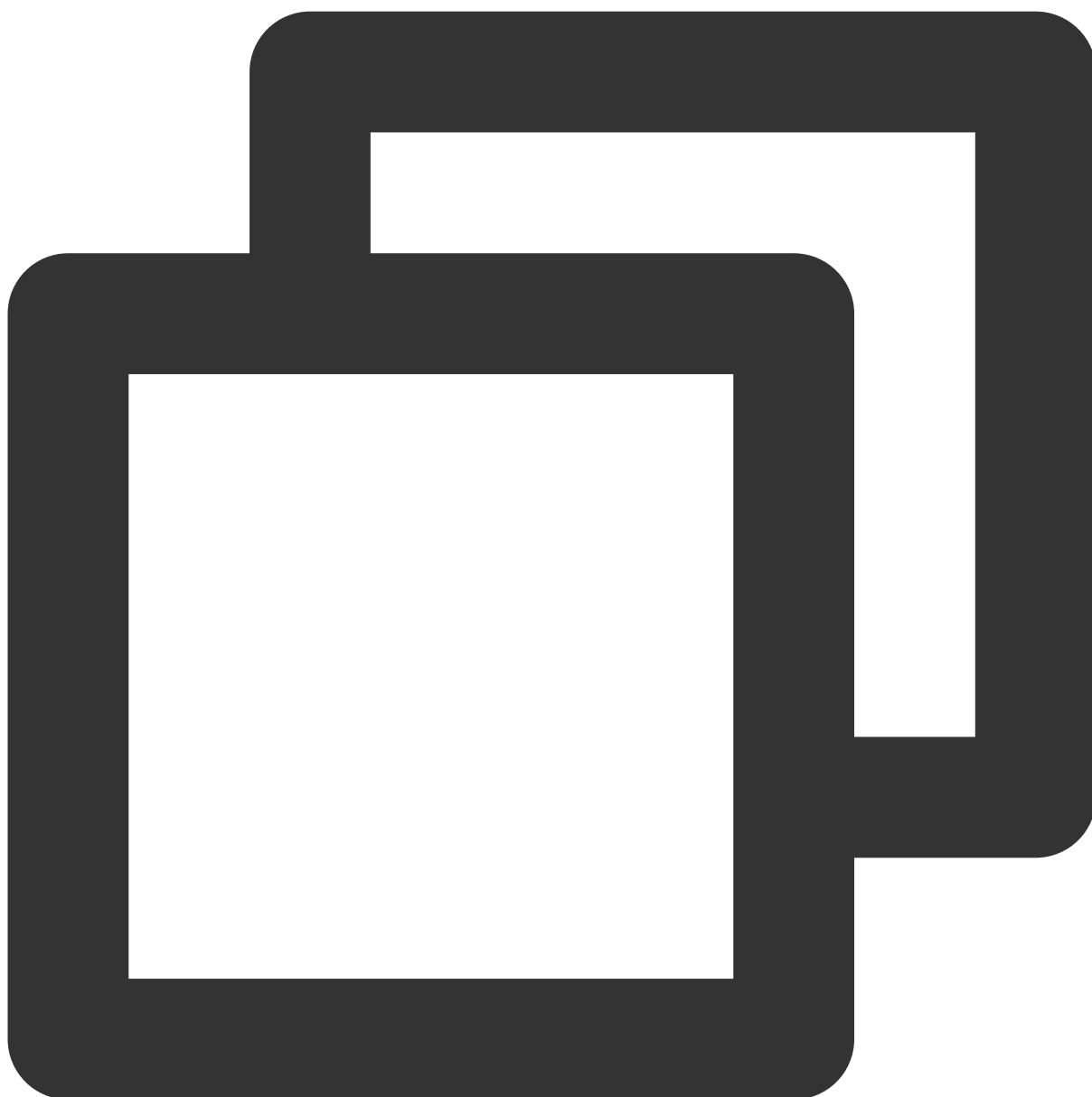
2. 将下载的小程序包拷贝到自定义的 `assets` 目录，离线小程序需严格遵守命名规则，不能随意修改。

说明：

离线小程序命名规则：`{miniAppId}_{miniAppVersion}.apkg`。



3. 在 SDK 初始化配置中指定上一步放置离线小程序的目录。



```
[[TMFMiniAppSDKManager sharedInstance] setOffLineApkgsPath:[[[NSBundle mainBundle]
```

注意事项

离线小程序也需要走正常的小程序绑定应用和发布流程，只有上线状态的小程序才可下载离线包。

离线小程序适用新版发布、版本回退、下架等版本管理逻辑，当线上版本与预置版本不一致时，客户端会正常拉取线上版本。

当小程序被下架后，预置在 **App** 中的对应离线小程序也无法正常使用。

自定义小程序能力

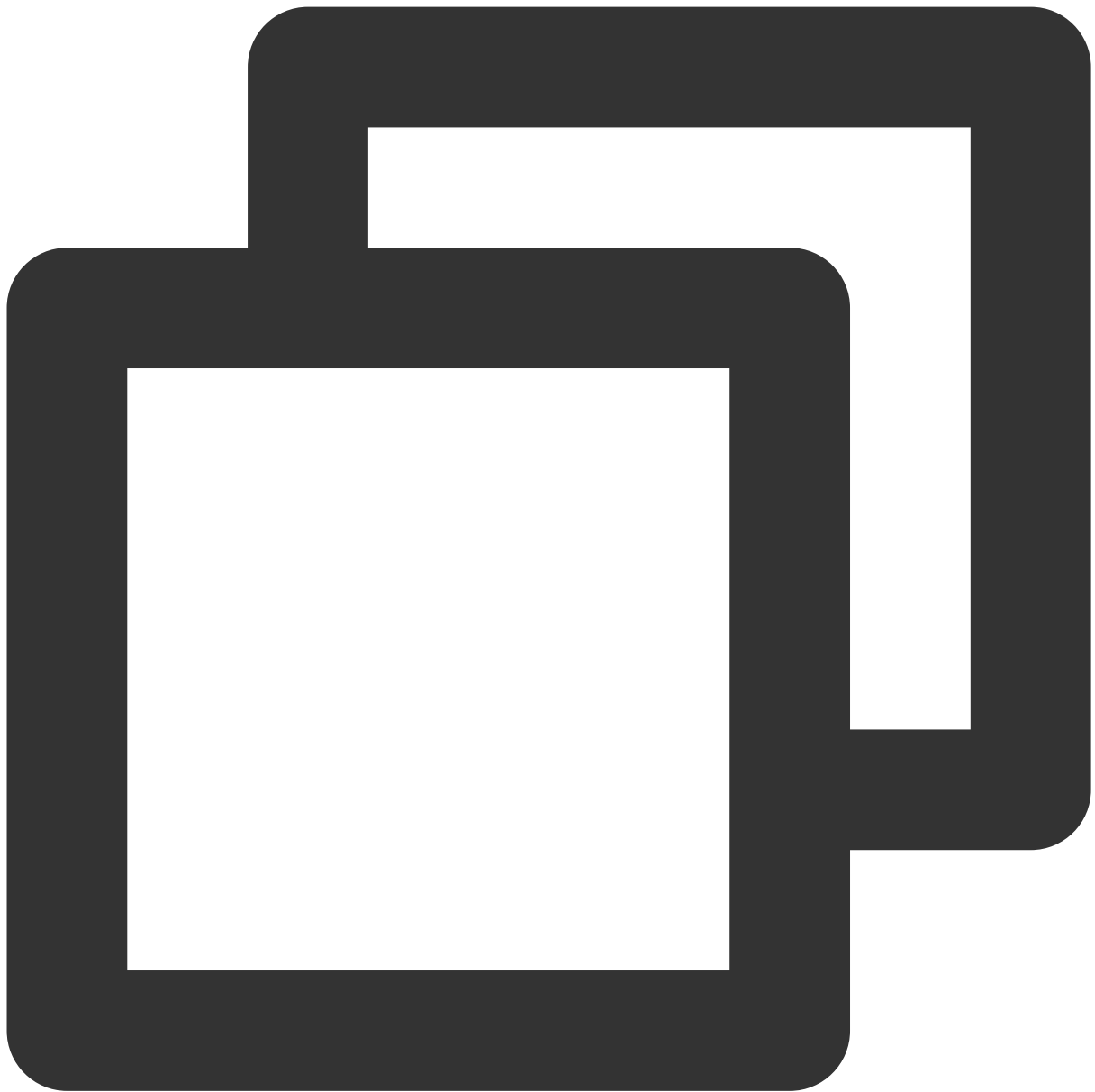
自定义小程序 API

最近更新时间：2024-07-03 18:03:50

TCMPP SDK 引擎提供扩展机制，允许宿主 App 自定义 API 供小程序调用。

实现步骤

1. 自定义一个类，并导入 TMAExternalJSPlugin。



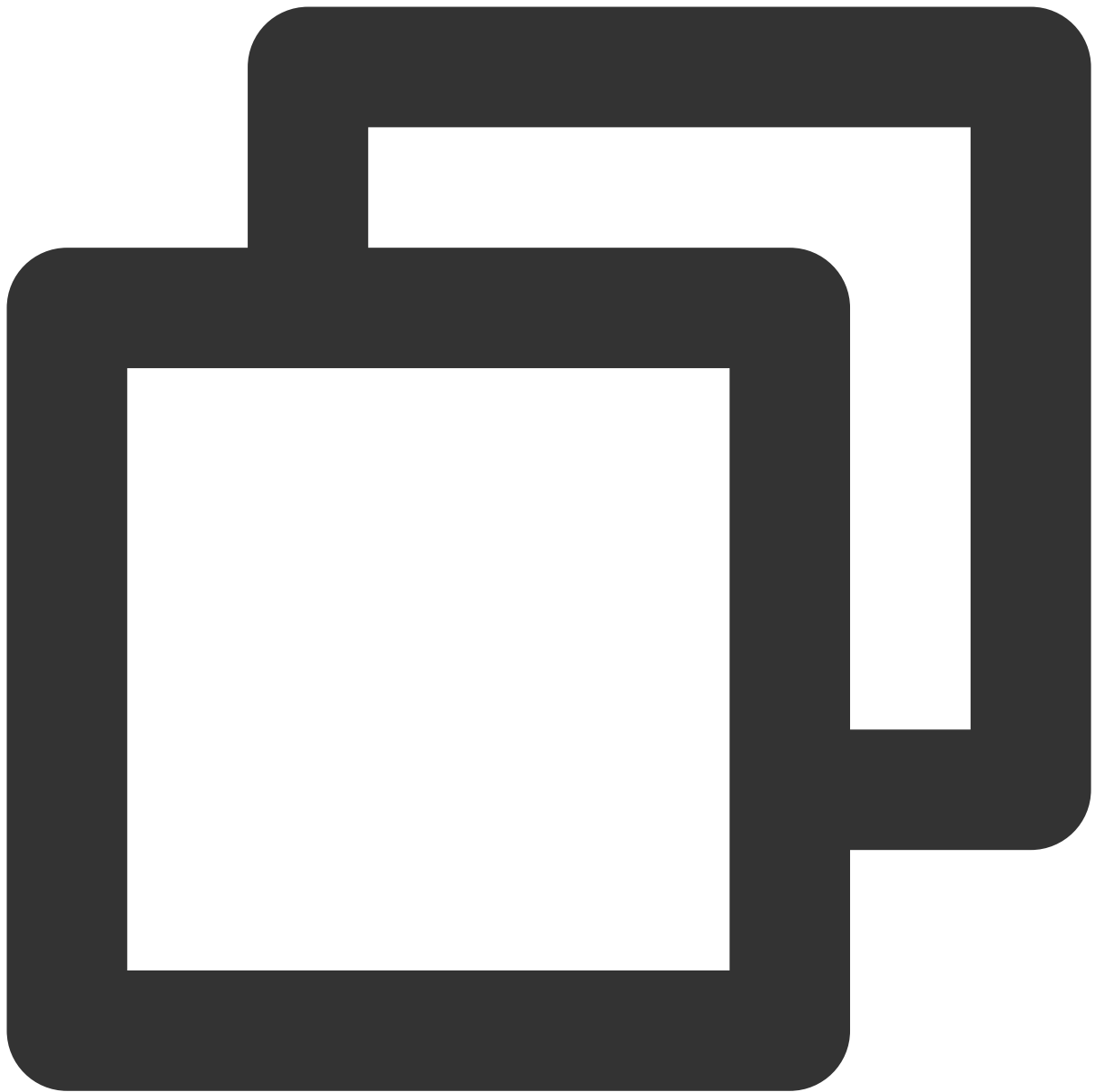
```
#import "TMAExternalJSPlugin.h"

@interface NativePluginTest : NSObject

@end
```

2. 声明 TMA_REGISTER_EXTENAL_JSPLUGIN 并通过 TMAExternalJSAPI_IMP() 添加自定义 API。

参考用例



```
#import "NativePluginTest.h"
#import "TMAExternalJSPlugin.h"
#import "TMFMiniAppInfo.h"

@implementation NativePluginTest

TMA_REGISTER_EXTENAL_JSPLUGIN;

//自定义同步 api
TMAExternalJSAPI_IMP(testSync) {
    TMFMiniAppInfo *appInfo = context.tmfAppInfo;
```

```
NSMutableDictionary *data = params[@"data"];

NSLog(@"***** invokeNativePlugin testSync,appId:%@,data is %@",appInfo.a

TMAExternalJSPluginResult *pluginResult = [TMAExternalJSPluginResult new];
pluginResult.result = @{};
return pluginResult;
}

TMAExternalJSAPI_IMP(test) {
    TMFMiniAppInfo *appInfo = context.tmfAppInfo;
    NSMutableDictionary *data = params[@"data"];

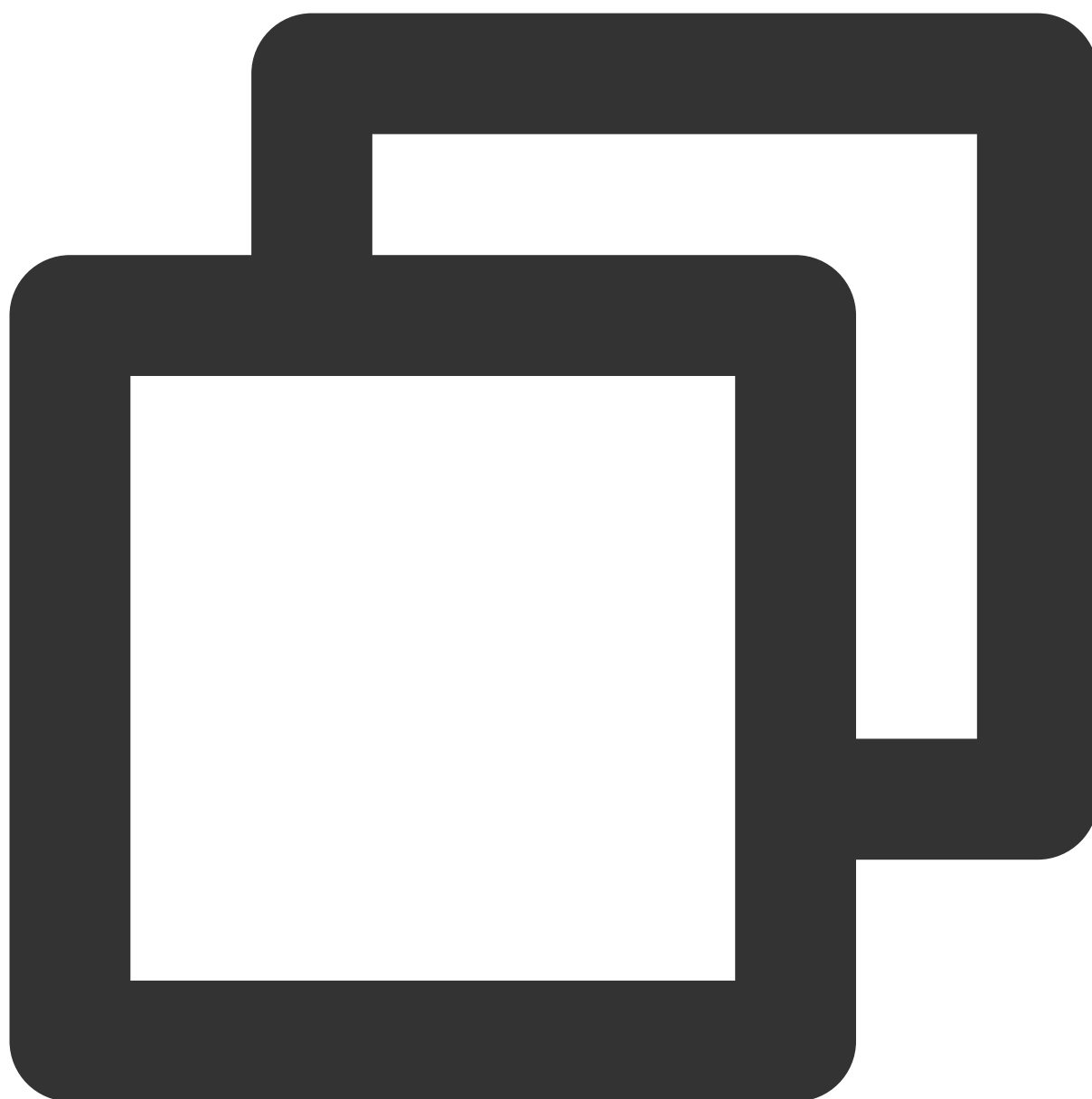
    NSLog(@"***** invokeNativePlugin test,appId:%@,data is %@",appInfo.appId

    //异步处理，在异步回调中把结果返回给小程序 async callback
    //{
    //    TMAExternalJSPluginResult *pluginResult = [TMAExternalJSPluginResult new];
    //    pluginResult.result = @{@"result" : result.data};
    //    [context doCallback:pluginResult];
    // }

    return nil;
}

@end
```

在小程序中可以这样使用。



```
//异步api调用
var opts = {
  api_name: 'test',
  success: function(res) {},
  fail: function(res) {},
  complete: function(res) {},
  data: { // 入参
    name : 'kka',
    age : 22
  }
}
```

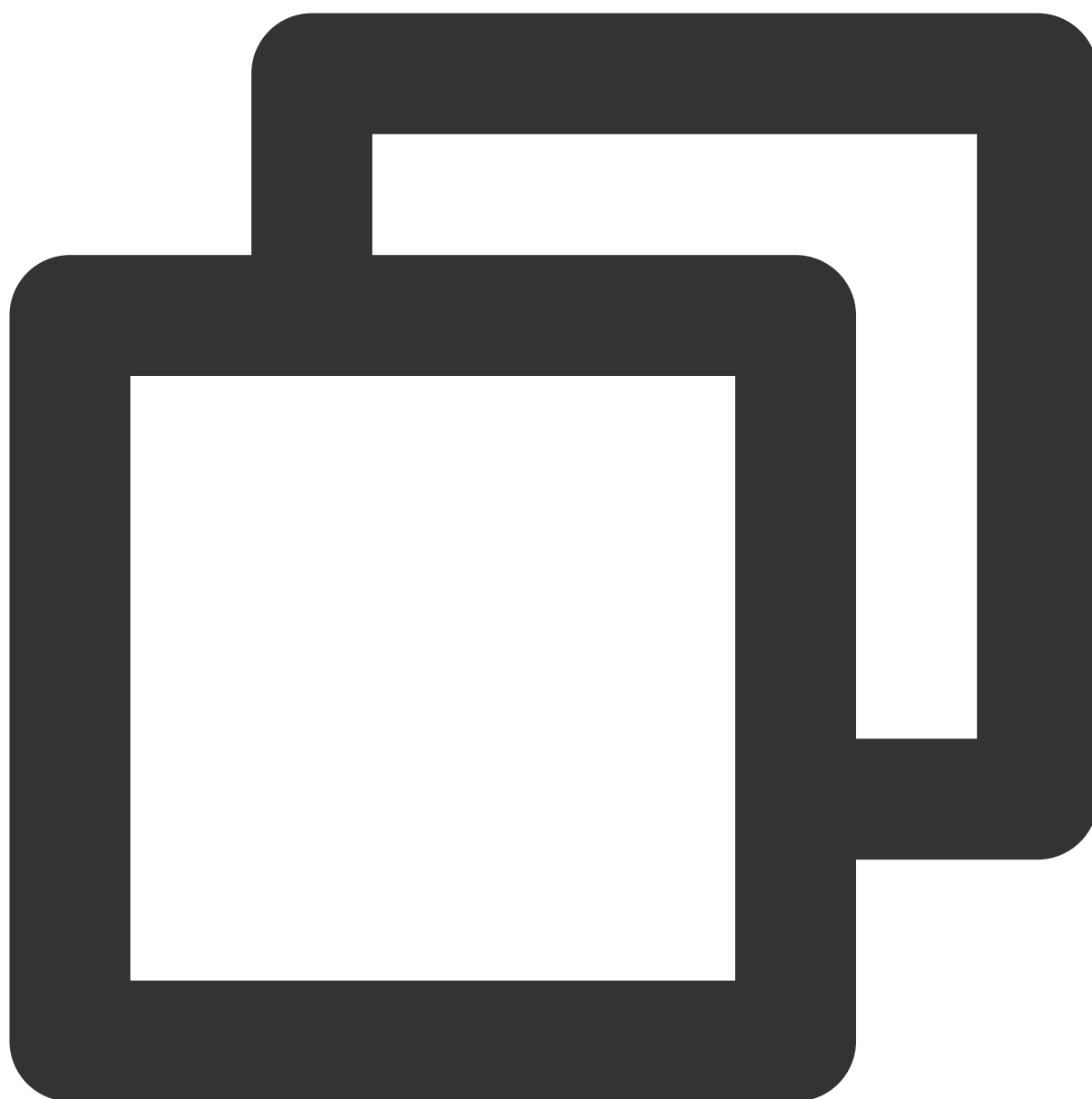
```
wx.invokeNativePlugin(opts);

//同步api调用
var opts = {
  api_name: 'testSync',
  sync:true
}
var rst = wx.invokeNativePlugin(opts);
```

进阶使用

自定义 api 支持在终端 app 配置文件的方式进行配置，在小程序中通过直接调用 `wx.api` 的方式来调用。

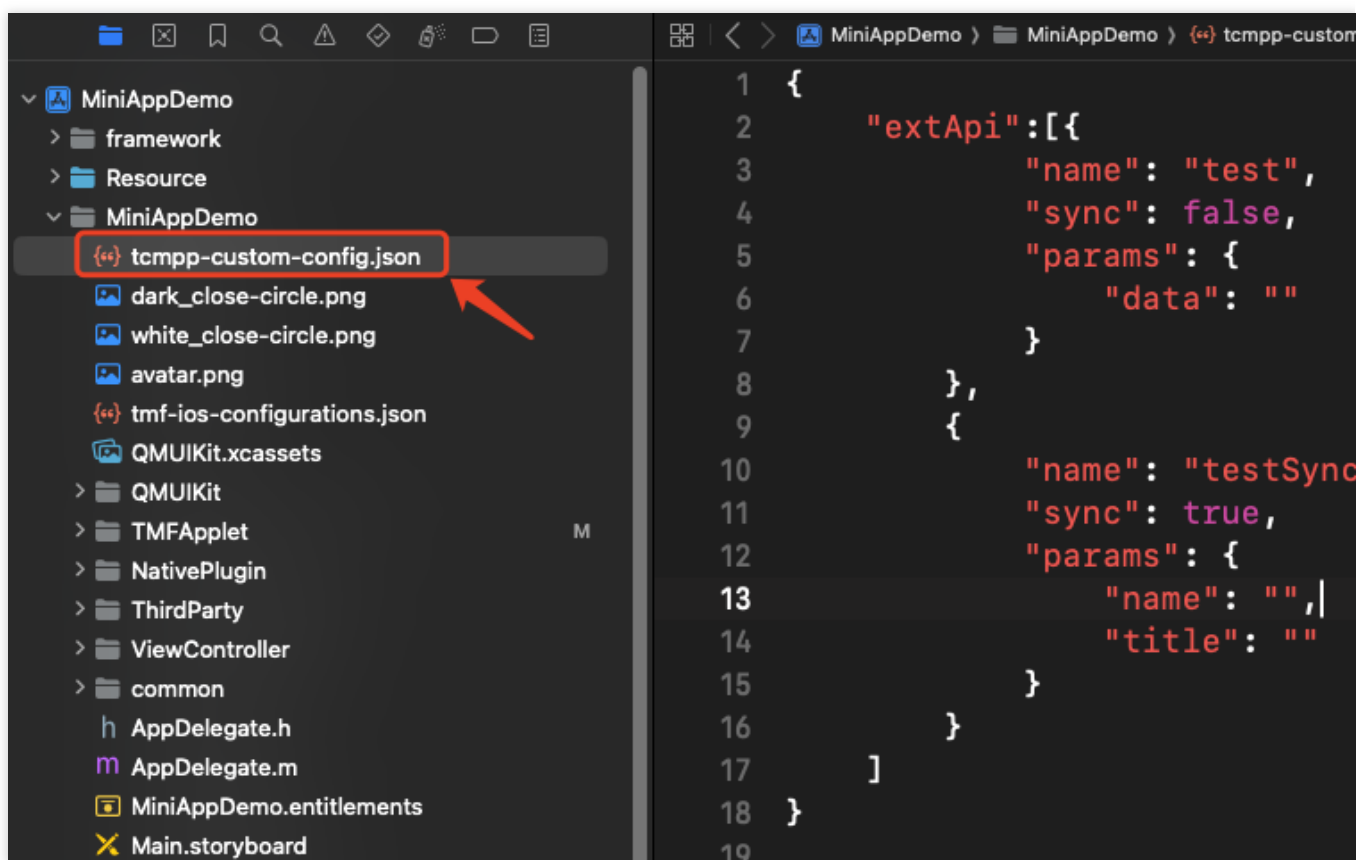
1. 将 app 中实现的配置文件统一放在 `tcmpp-custom-config.json` 中，参考内容如下：



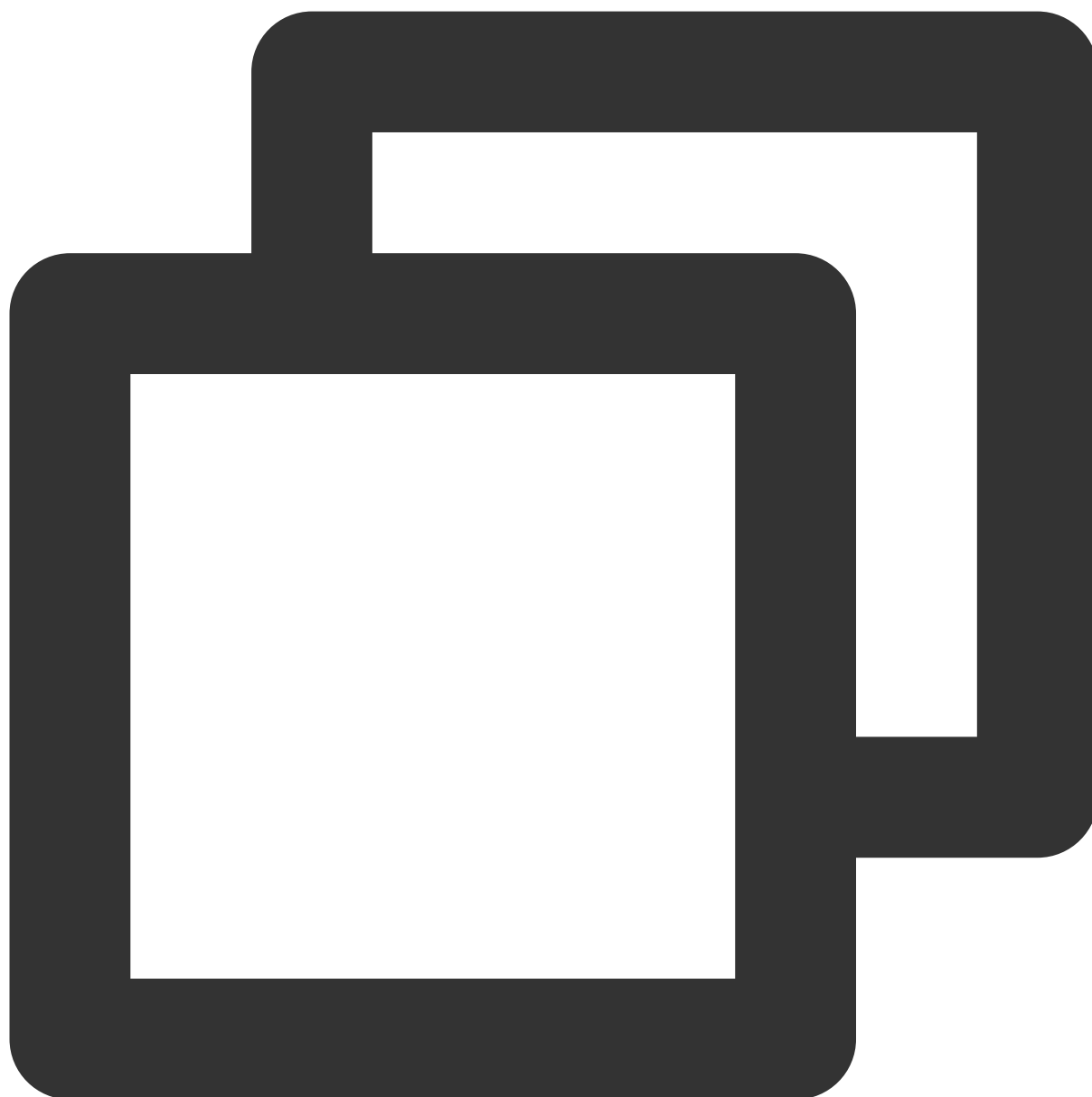
```
{
  "extApi": [{
    "name": "test",
    "sync": false,
    "params": {
      "data": ""
    }
  },
  {
    "name": "testSync",
    "sync": true,
```

```
        "params": {
          "name": "",
          "title": ""
        }
      }
    ]
  }
}
```

2. 把 tcmpp-custom-config.json 放入 iOS 工程中：



3. 在小程序中直接用 wx.test() 的方式调用。



```
//异步api调用
var opts = {
  success: function(res) {},
  fail: function(res) {},
  complete: function(res) {},
  data: {
    name : 'kka',
    age : 22
  }
}
wx.test(opts);
```



```
//同步api调用  
var rst = testSync(opts);
```

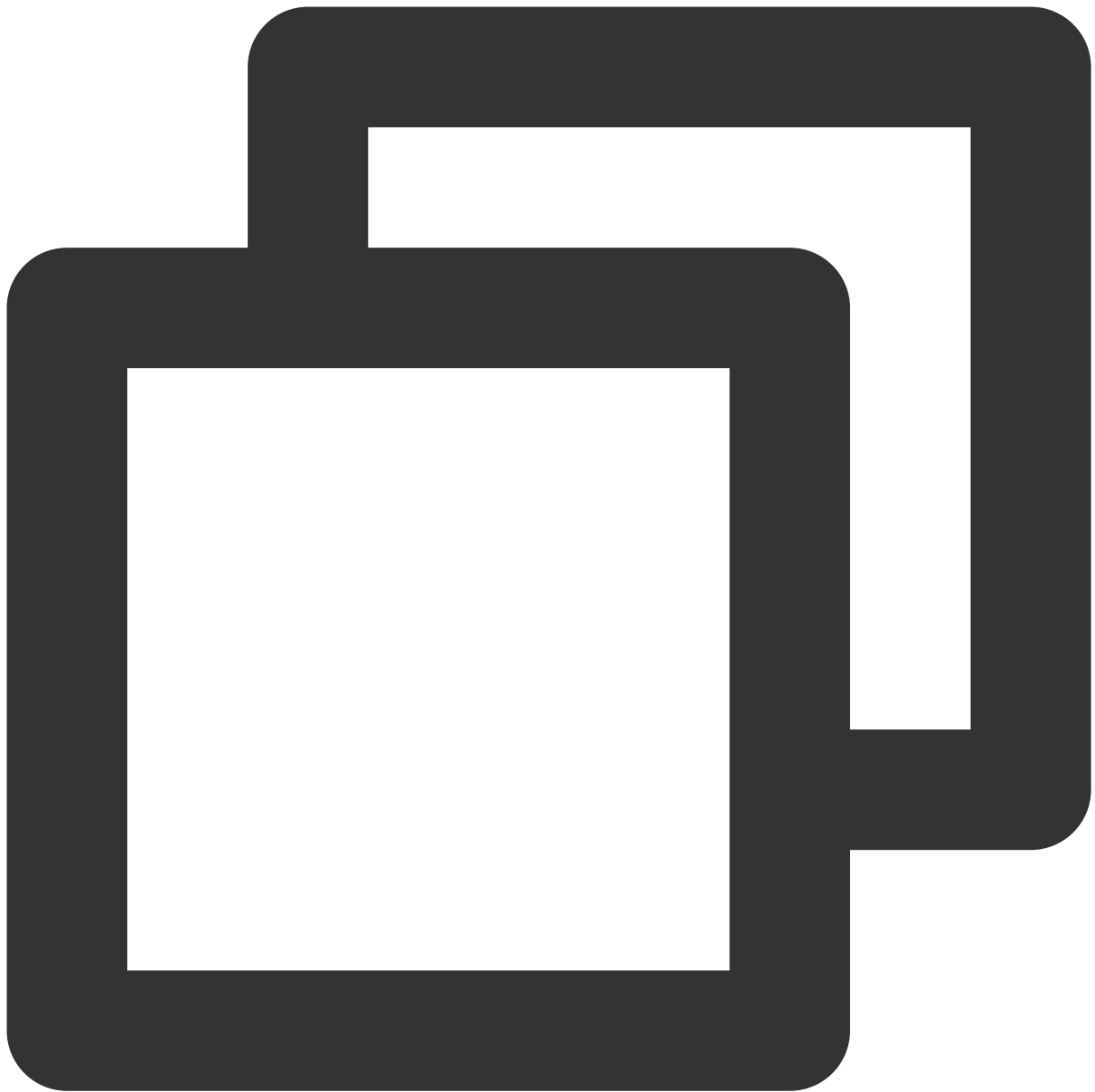
自定义小程序视图组件

最近更新时间：2024-07-03 18:04:02

使用自定义原生组件

TCMPP 支持自定义 UI 组件，小程序端的自定义组件为 `<external-element>`。如果您想使用自定义 UI 组件，请遵循以下步骤：

1. 客户端接入 SDK 后新建继承自 `UIView` 的类，例如 `QMATestView`，导入 `"TMAExternalJSPlugin.h"` 文件，使 `QMATestView` 遵循 `"TMAExternalElementView"` 协议。
2. 调用 `TMARegisterExternalElement` 方法，注册 `QMATestView` 类为 `maTestView`。
3. 实现 `"TMAExternalElementView"` 协议中的 `createWithParams` 和 `operateWithParams` 方法。



```
#import "QMATestView.h"
#import "TMAExternalJSPlugin.h"

@interface QMATestView () <TMAExternalElementView>

@end

@implementation QMATestView {
    UILabel *_textLabel;
    UIButton *_clickButton;
    id<TMAExternalJSContextProtocol> _context;
```

```

}

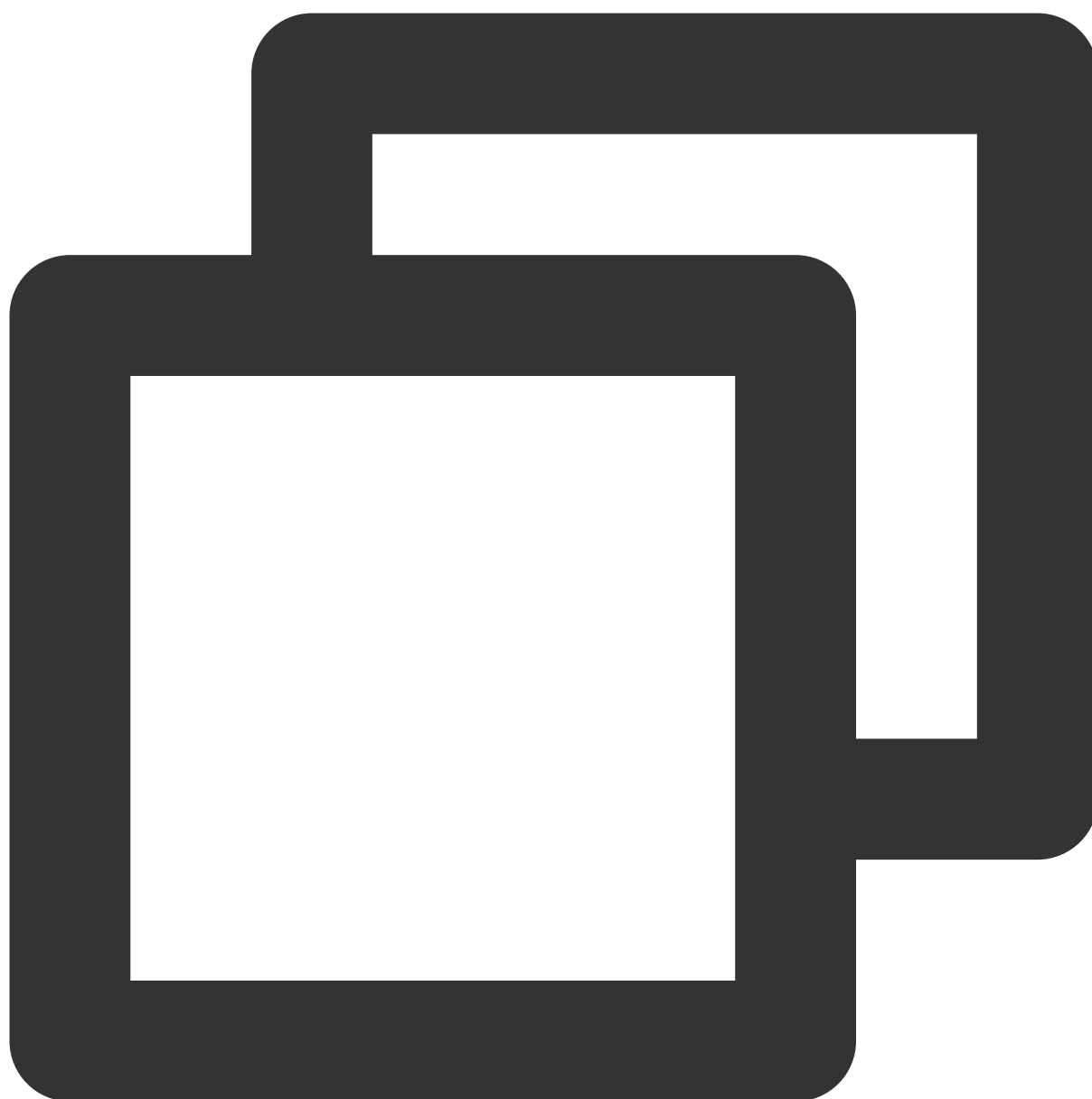
TMARegisterExternalElement (maTestView);
+ (UIView *)createWithParams:(NSDictionary *)params context:(id<TMAExternalJSContext>)context {
    QMATestView *testView = [[QMATestView alloc] initWithFrame:CGRectZero];
    NSDictionary *testViewParams = QQ_Dict_DictValue(params, @"params");
    [testView setText:QQ_Dict_StringValue(testViewParams, @"text")];
    testView->_context = context;
    return testView;
}

//接收小程序端的调用事件进行处理
- (void)operateWithParams:(NSDictionary *)param context:(id<TMAExternalJSContext>)context {
    NSDictionary *data = QQ_Dict_DictValue(param, @"data");
    NSDictionary *params1 = QQ_Dict_DictValue(data, @"params1");
    NSInteger age = [QQ_Dict_NumberValue(params1, @"age") integerValue];
    NSString *name = QQ_Dict_StringValue(params1, @"name");
    qq_weakify(self);
    [MAUtils executeOnThread:[NSThread mainThread] block:^(
        qq_strongify(self);
        if (self) {
            self->_textLabel.text = [NSString stringWithFormat:@"name = %@ , age = %ld", name, age];
            // 把结果返回给小程序端
            TMAExternalJSPluginResult *result = [TMAExternalJSPluginResult new];
            result.result = @{@"result":@"success"};
            [context doCallback:result];
        }
    }];
}

```

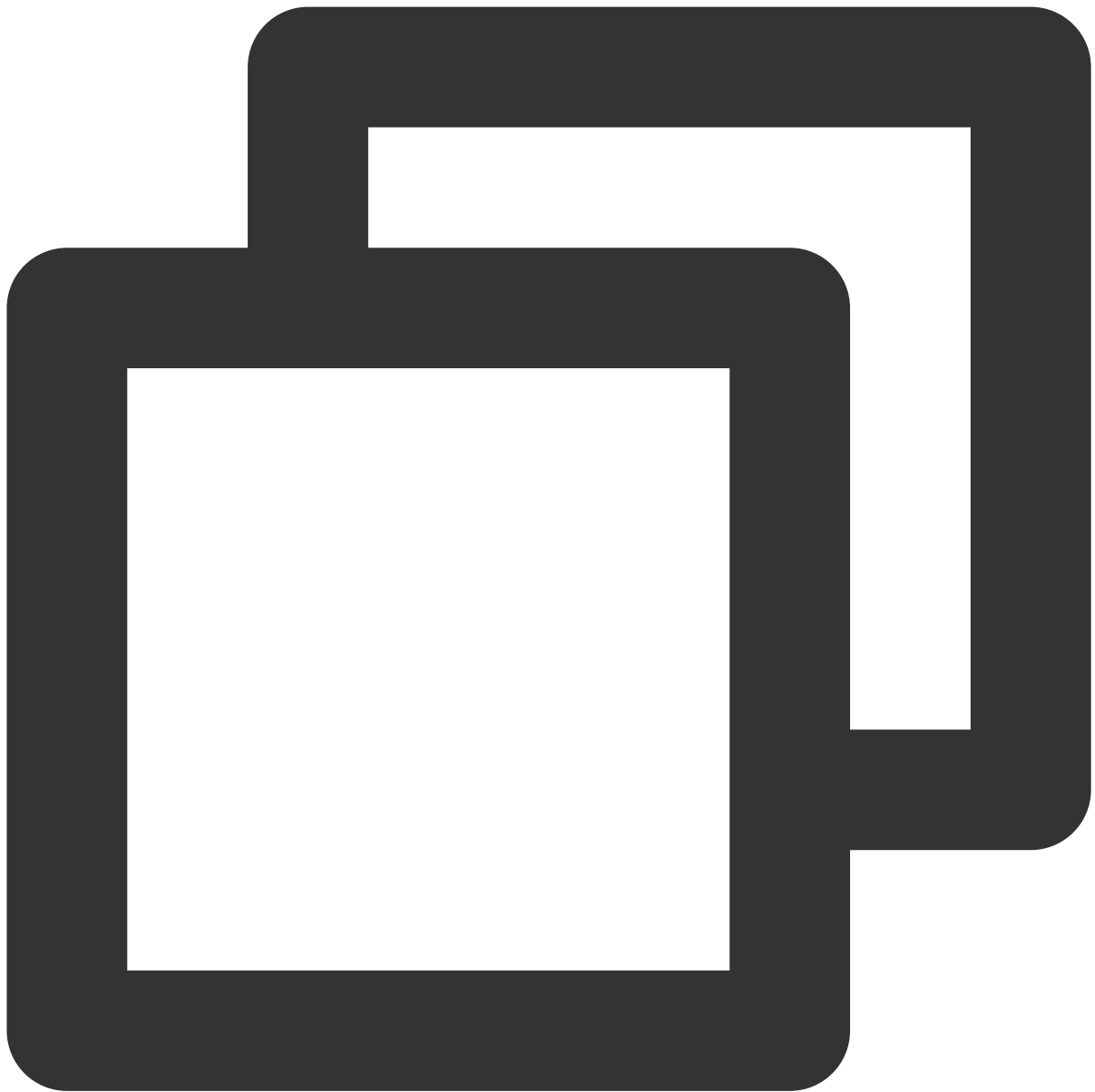
发送事件给小程序端

在自定义原生组件里如果想发送事件给小程序端，先在 `createWithParams:context:` 方法中记录 `context`：



```
_context = context;
```

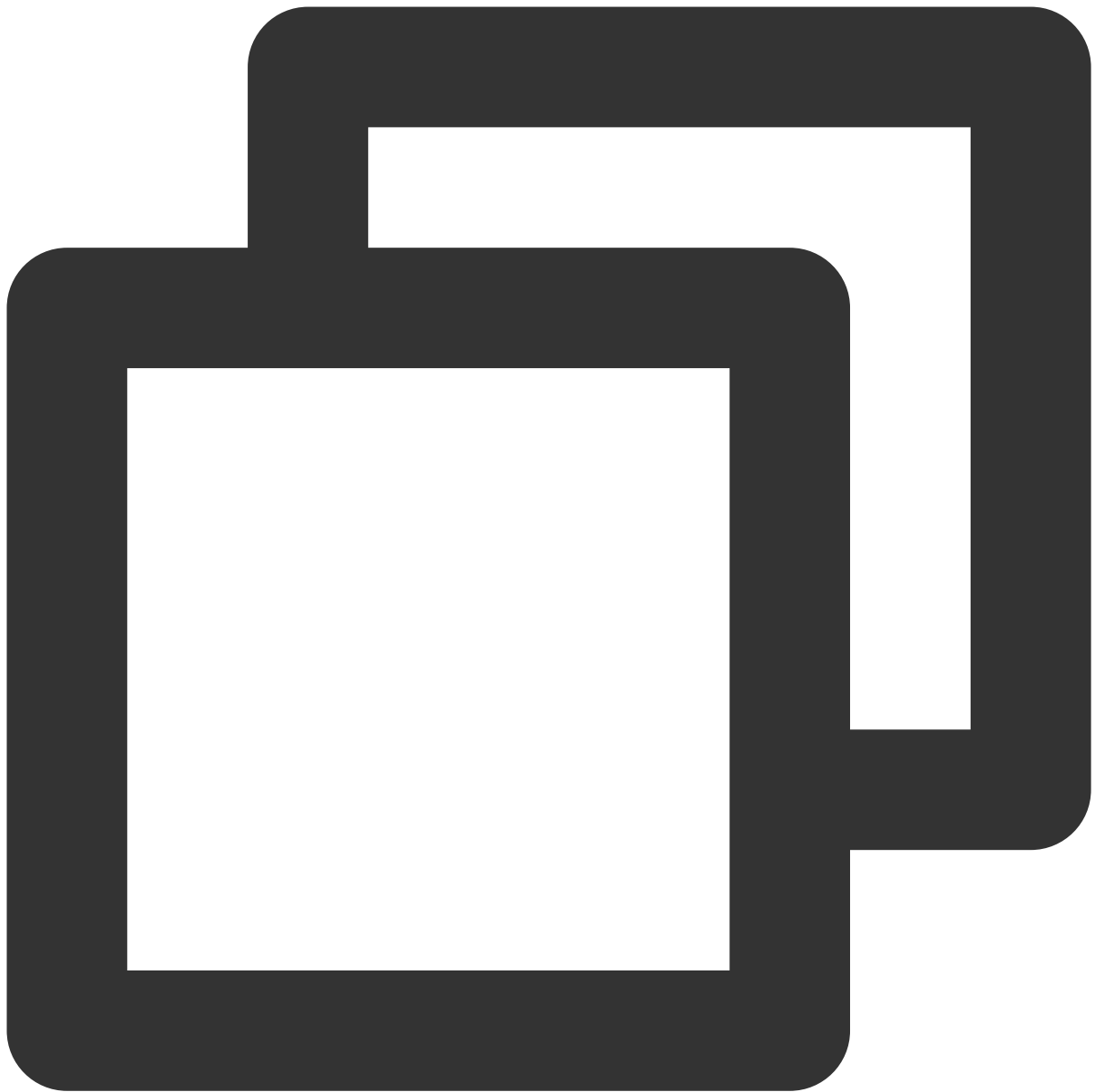
发送事件时这样写：



```
- (void)onClickButton:(UIButton *)button {
    _textLabel.text = @"What do you want me to do";
    // 组装数据 发送事件
    NSString *data = [MAUtils JSONStringify:@{@"externalElementId":_elementId,@"typ
    [_context doSubscribe:kTMAOnExternalElementEvent data:data];
}
```

小程序端使用

1. 在小程序 wxml 中引入：

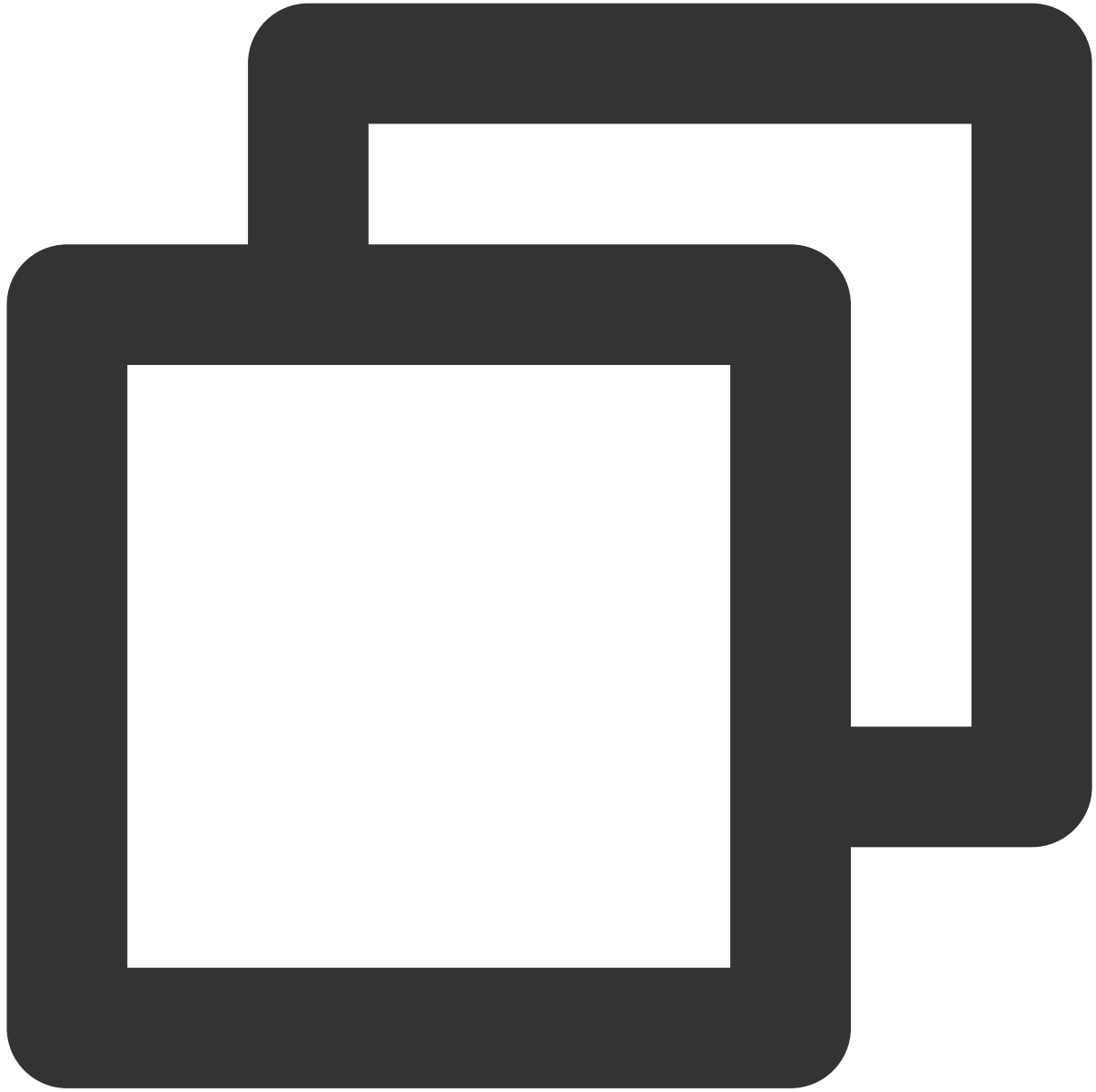


```
<external-element
  id="comp1"
  type="maTestView"
  _insert2WebLayer
  style="width: 200px;height: 100px;"
  bindexternalelementevent="handleEvent"
></external-element>
```

说明：

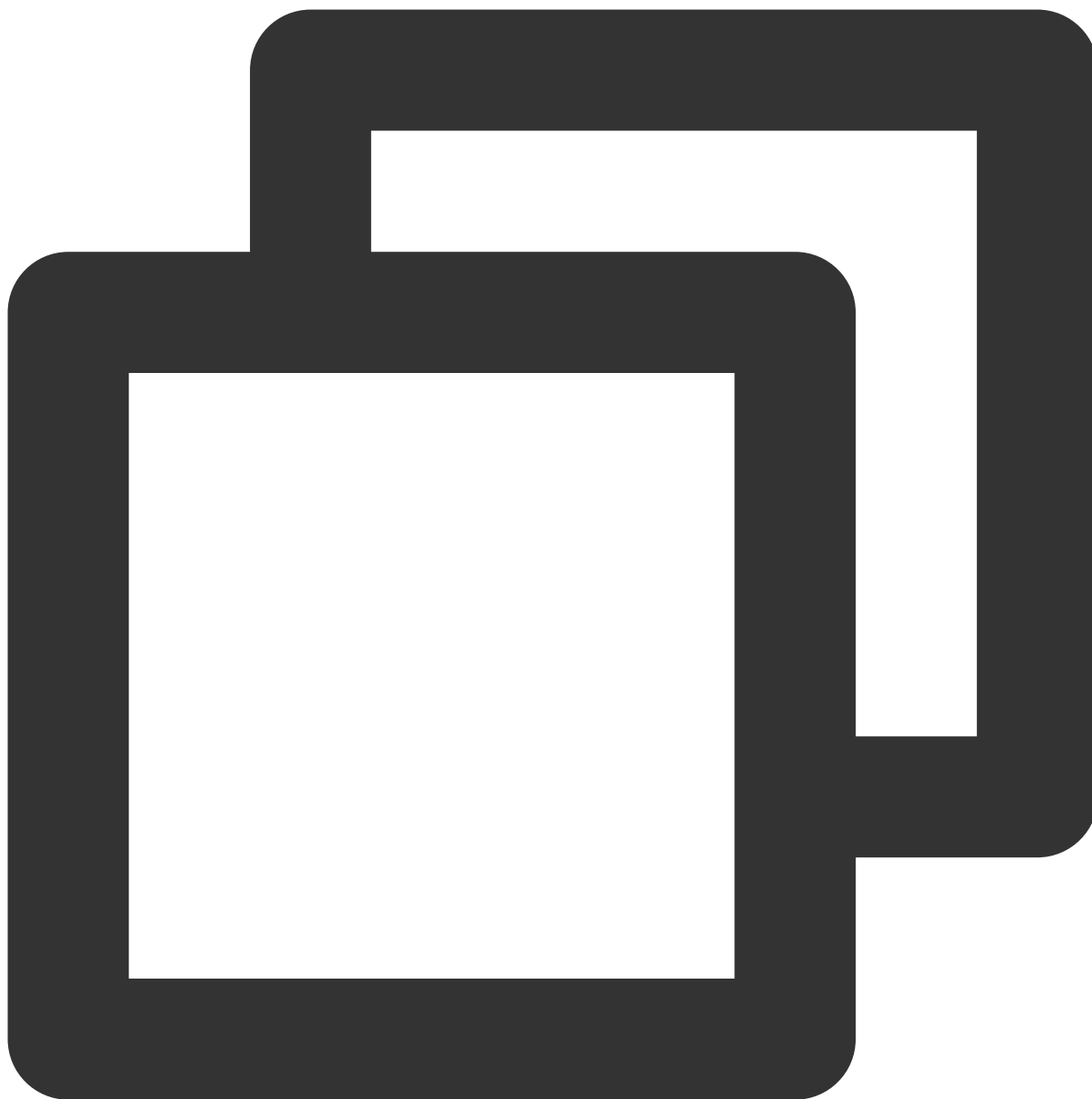
type 需与 native 端约定，如非同层则不设置 _insert2WebLayer 属性。

`bindexternalelementevent` 可监听 native 传递的操作，回调参数包括：



```
{
  target,
  currentTarget,
  timeStamp,
  touches,
  detail, // native传递的参数
}
```


2. 小程序 js 中创建实例。

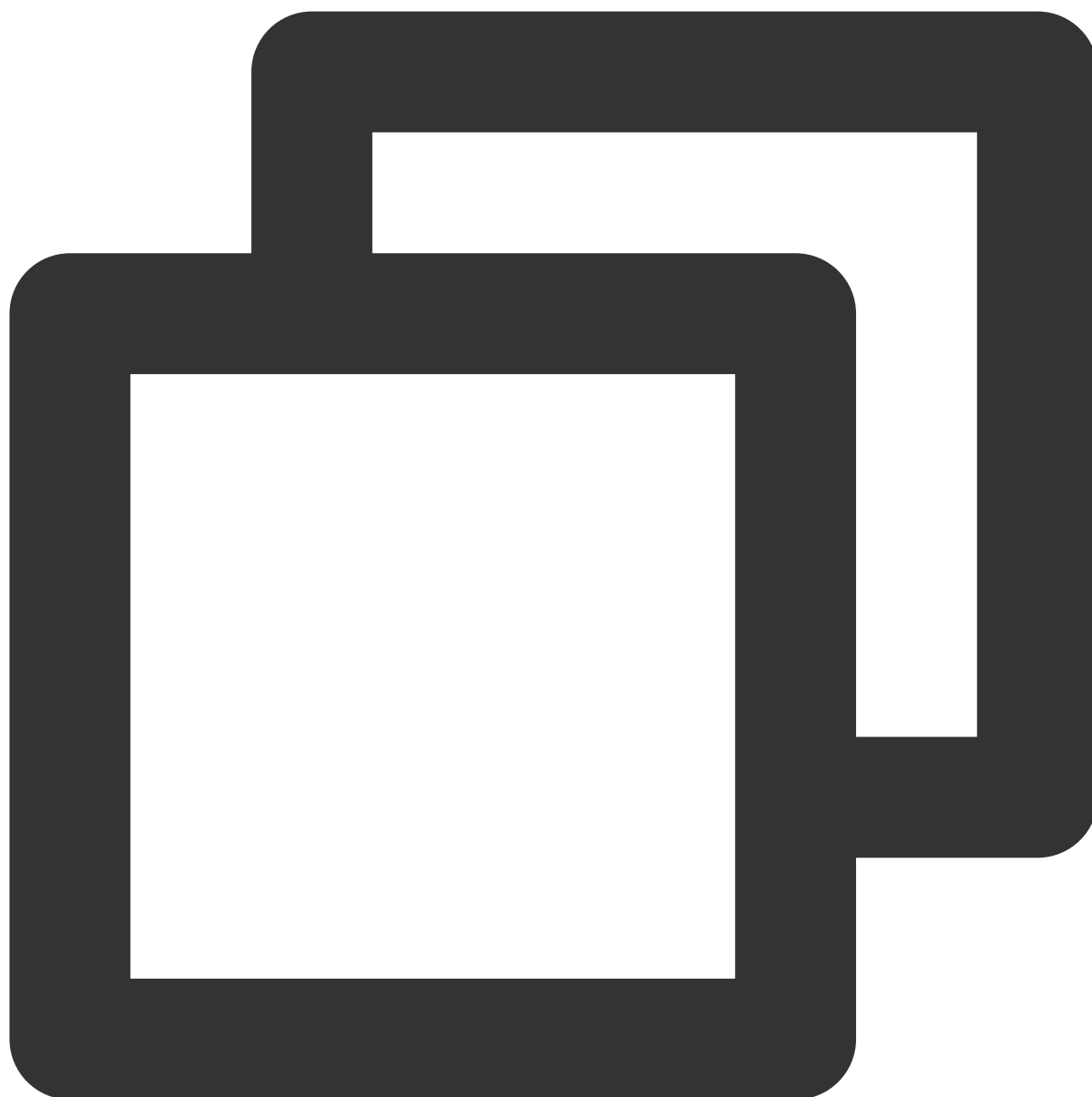


```
this.ctx = wx.createExternalElementContext('comp1');
```

说明：

`wx.createExternalElementContext` 参数为 `wxml` 中元素 `id`。

3. 在小程序中调用实例方法：



```
this.ctx.call({
  params1: {
    name: 'name1',
    age: 11
  },
  params2: {
    name: 'name2',
    age: 22
  },
  success: (e) => {
    console.log('===operate success===', e)
```

```
},
fail: (e) => {
  console.log('====operate fail====', e)
},
complete: (e) => {
  console.log('====operate complete====', e)
}
})
```

说明：

实例提供 `call` 方法，`success`、`fail`、`complete` 回调；通过参数区分调用方法，基础库会调用 `invoke` 方法透传参数到 `native`。

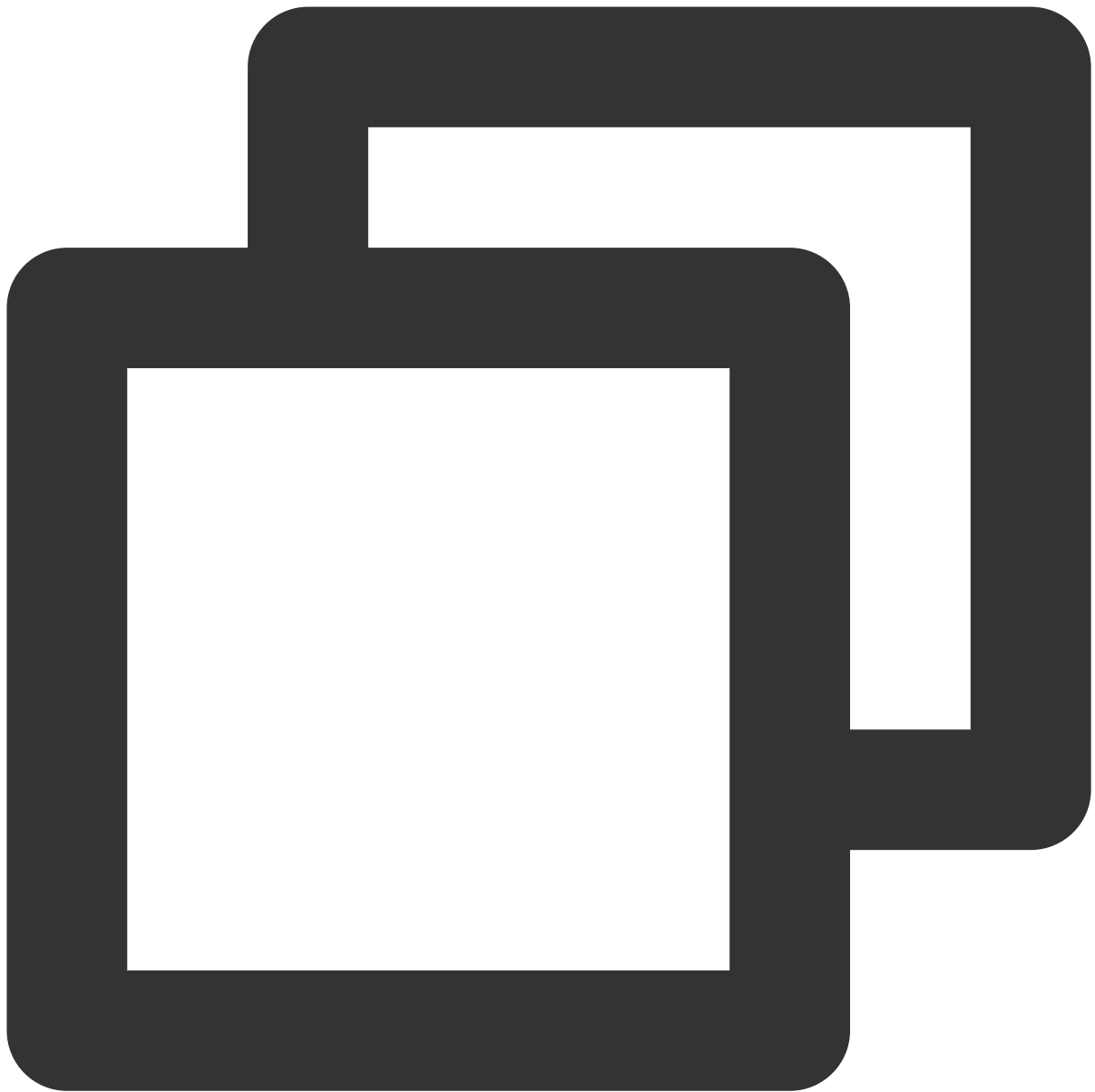
定制 SDK 能力介绍

最近更新时间：2024-07-03 18:15:00

为了丰富使用场景，小程序 SDK 对外开放了一部分接口，这部分接口可以由宿主应用定制实现，以提供宿主应用特色的能力给小程序使用。

前提条件

实现小程序 SDK 的接口定制，需要创建一个类并遵循 `TMFMiniAppSDKDelegate` 协议：



```
#import "TMFMiniAppSDKDelegate.h"

@interface MIniAppDemoSDKDelegateImpl : NSObject <TMFMiniAppSDKDelegate>

@end
```

前提条件的步骤完成后，可以按照相关文档进行小程序 SDK 接口进行自定义了。

相关文档

[定制小程序UI](#)

[定制小程序信息接口](#)

[定制分享能力](#)

[定制用户属性](#)

[日志与事件上报](#)

[开放接口](#)

[其他](#)

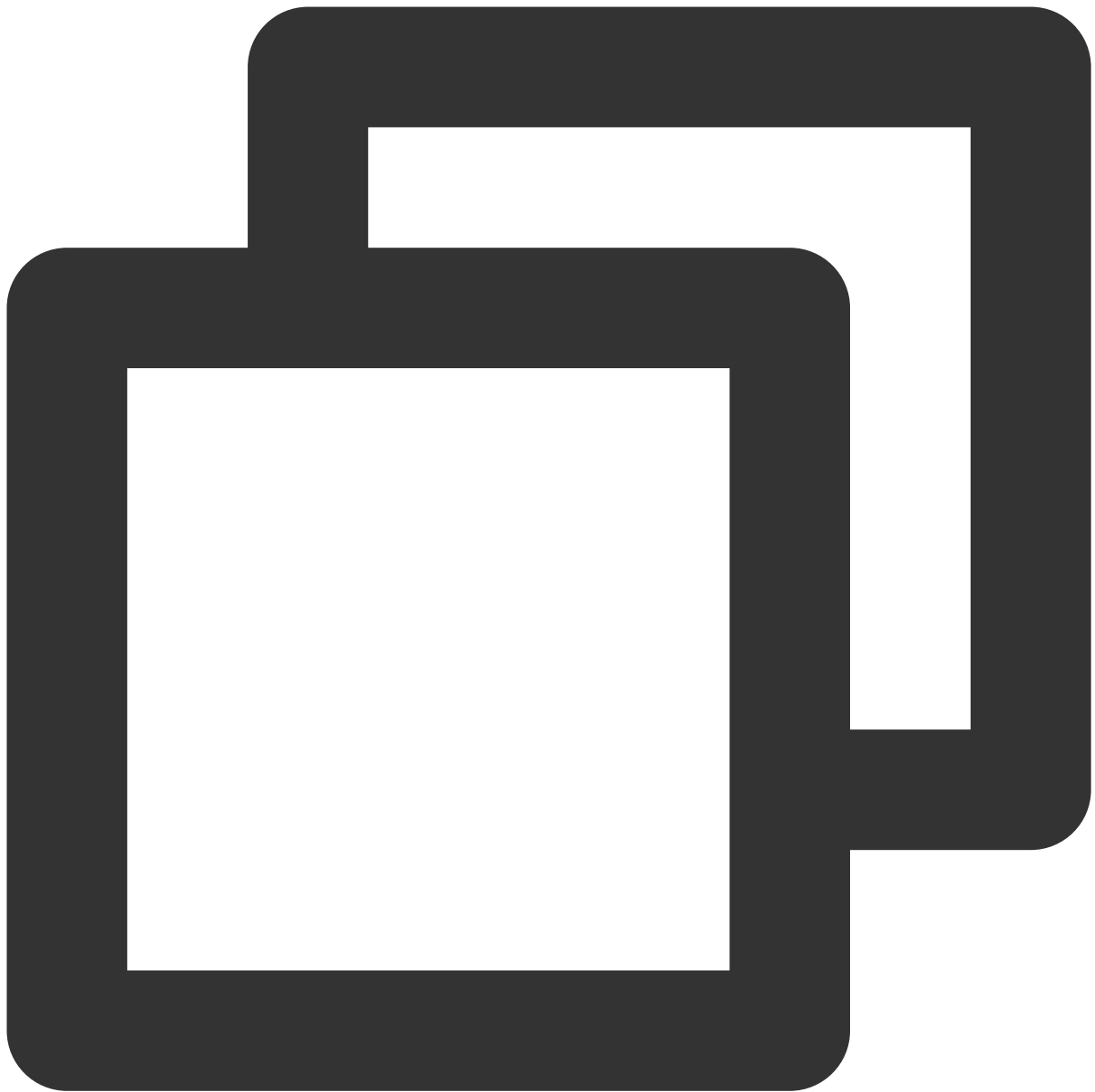
定制小程序 UI

最近更新时间：2024-07-03 18:14:51

小程序 SDK 开放了对小程序的部分原生UI效果的自定义能力，开发者可以按照下面的规范进行配置。

设置加载页 UI

TCMPP 小程序引擎支持宿主 App 重新定义小程序加载时的 loading 页面，来代替 SDK 内容默认的加载页，具体实现是通过实现 `TMFMiniAppSDKDelegate` 协议中的 `customLoadingViewWithAppInfo`，参考代码如下：



```
- (UIView *)customLoadingViewWithAppInfo:(TMFMiniAppInfo *)appInfo frame:(CGRect)frame {
    UIView *view = [[UIView alloc] initWithFrame:frame];
    //todo 设置具体的 view 相关内容

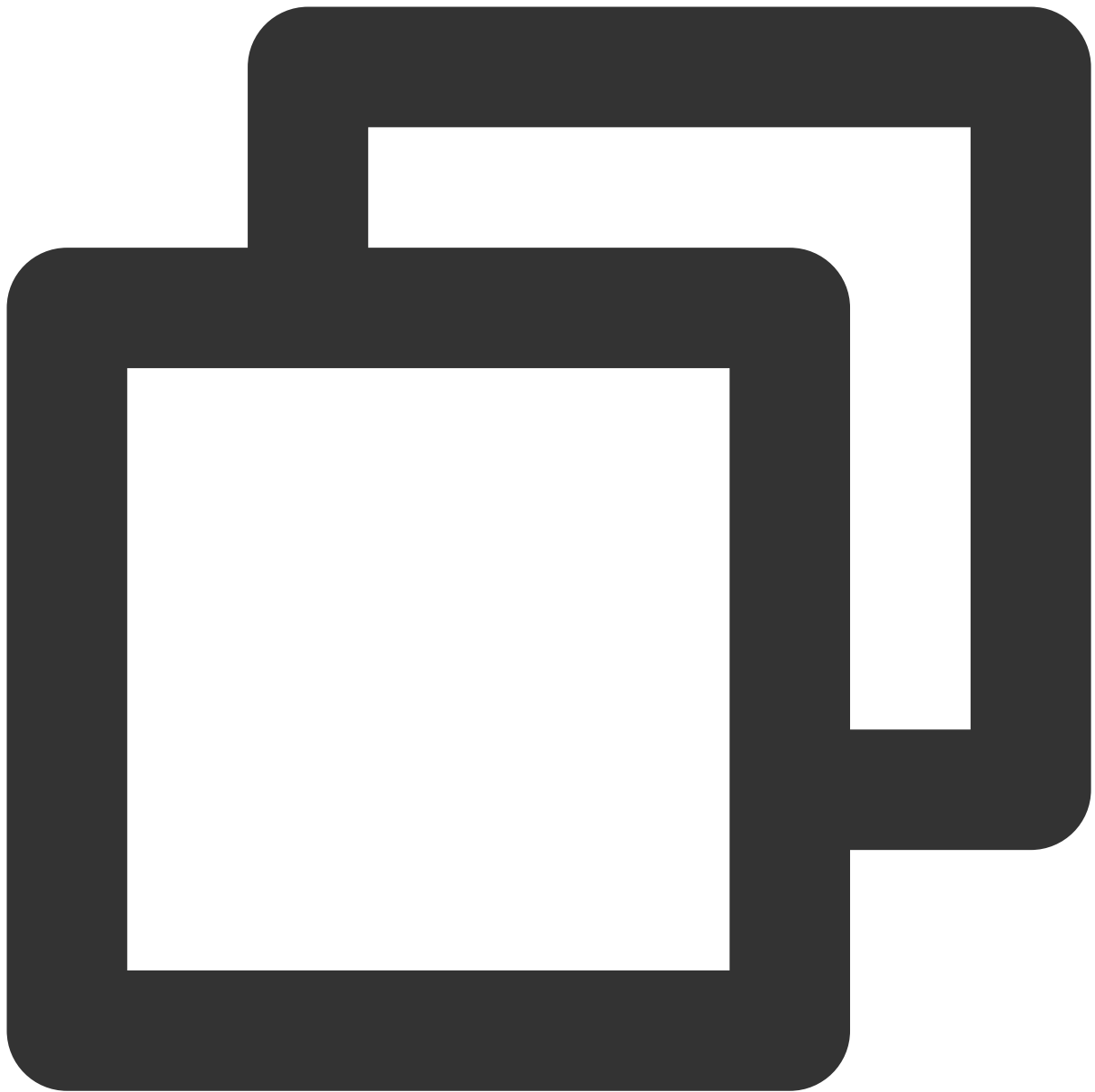
    return view;
}
```

设置小程序导航栏资源

TCMPP 小程序引擎支持宿主 App 重新定义小程序导航栏资源来替换默认资源，来实现自己的风格，具体实现是通过实现 TMFMiniAppSDKDelegate 协议中的 stringWithConfigKey，目前支持以下内容的设置：

Key	描述
TMA_SK_MINIAPP_CloseButton	关闭按钮 icon
TMA_SK_MINIAPP_CloseButtonDark	关闭按钮 icon 深色主题
TMA_SK_MINIAPP_HomeButton	主页按钮 icon
TMA_SK_MINIAPP_HomeButtonDark	主页按钮 icon 深色主题
TMA_SK_MINIAPP_BackButton	返回按钮 icon
TMA_SK_MINIAPP_BackButtonDark	返回按钮 icon 深色主题
TMA_SK_MINIAPP_MoreButton	更多按钮 icon
TMA_SK_MINIAPP_MoreButtonDark	更多按钮 icon 深色主题
TMA_SK_MINIAPP_RecordButton	录音按钮 icon
TMA_SK_MINIAPP_RecordButtonDark	录音按钮 icon 深色主题
TMA_SK_MINIAPP_MoreBackground	胶囊部分背景图片
TMA_SK_MINIAPP_MoreBackgroundDark	胶囊部分背景图片深色主题

示例代码：



```
- (NSString *)stringWithConfigKey:(NSString *)key {  
    //设置浅色模式下的关闭按钮  
    if([key isEqualToString:TMA_SK_MINIAPP_CloseButton]) {  
        return [[[NSBundle mainBundle] resourcePath] stringByAppendingPathComponent  
    } else if([key isEqualToString:TMA_SK_MINIAPP_CloseButtonDark]) {  
        return [[[NSBundle mainBundle] resourcePath] stringByAppendingPathComponent  
    }  
  
    return nil;  
}
```

设置小程序更多菜单

TCMPP 小程序引擎支持宿主 App 重新定义小程序导航栏上更多按钮弹出胶囊的处理，包括：

1. 在胶囊页面增加自定义的菜单项

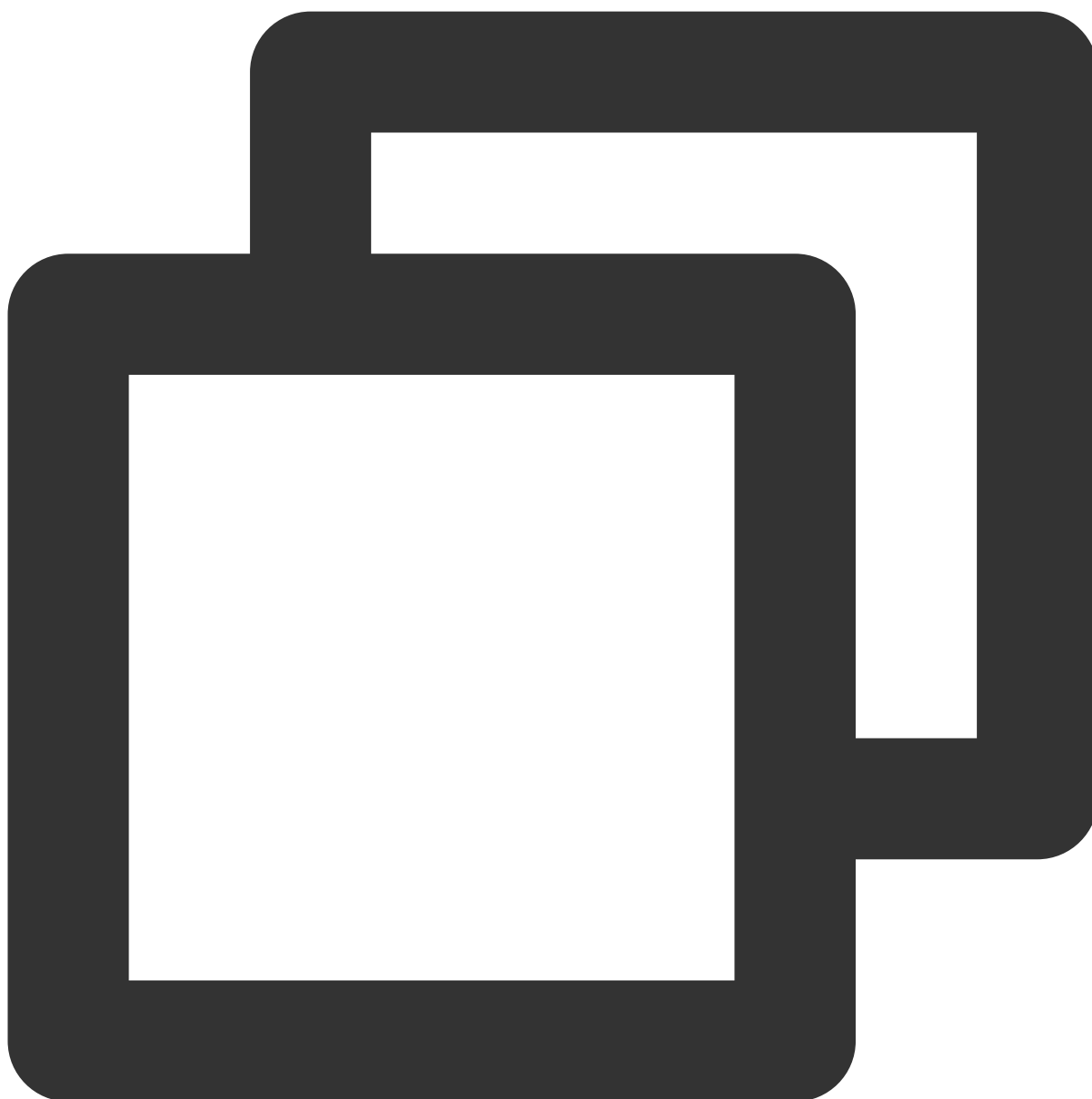
通过 `TMFMiniAppSDKDelegate` 协议中的 `customizedConfigForShare` 可以实现在胶囊视图中增加内容，增加的选项分为两类：

`MAUIDelegateShareViewTypeCustomizedShare`：分享类型，会走小程序内分享逻辑，在

`TMFMiniAppSDKDelegate` 中触发 `shareMessageWithMod` 实现的分享操作。

`MAUIDelegateShareViewTypeCustomizedAction`：自定义操作类型，可以自定义回调事件。

示例代码：



```
- (NSArray<TMASheetItemInfo *> *)customizedConfigForShare {
    NSMutableArray *arrays = [[NSMutableArray alloc] init];
    TMASheetItemInfo *item1 = [[TMASheetItemInfo alloc] initWithTitle:@"More sharin
    item1.icon = [UIImage imageNamed:@"icon_moreOperation_shareChat"];
    [arrays addObject:item1];

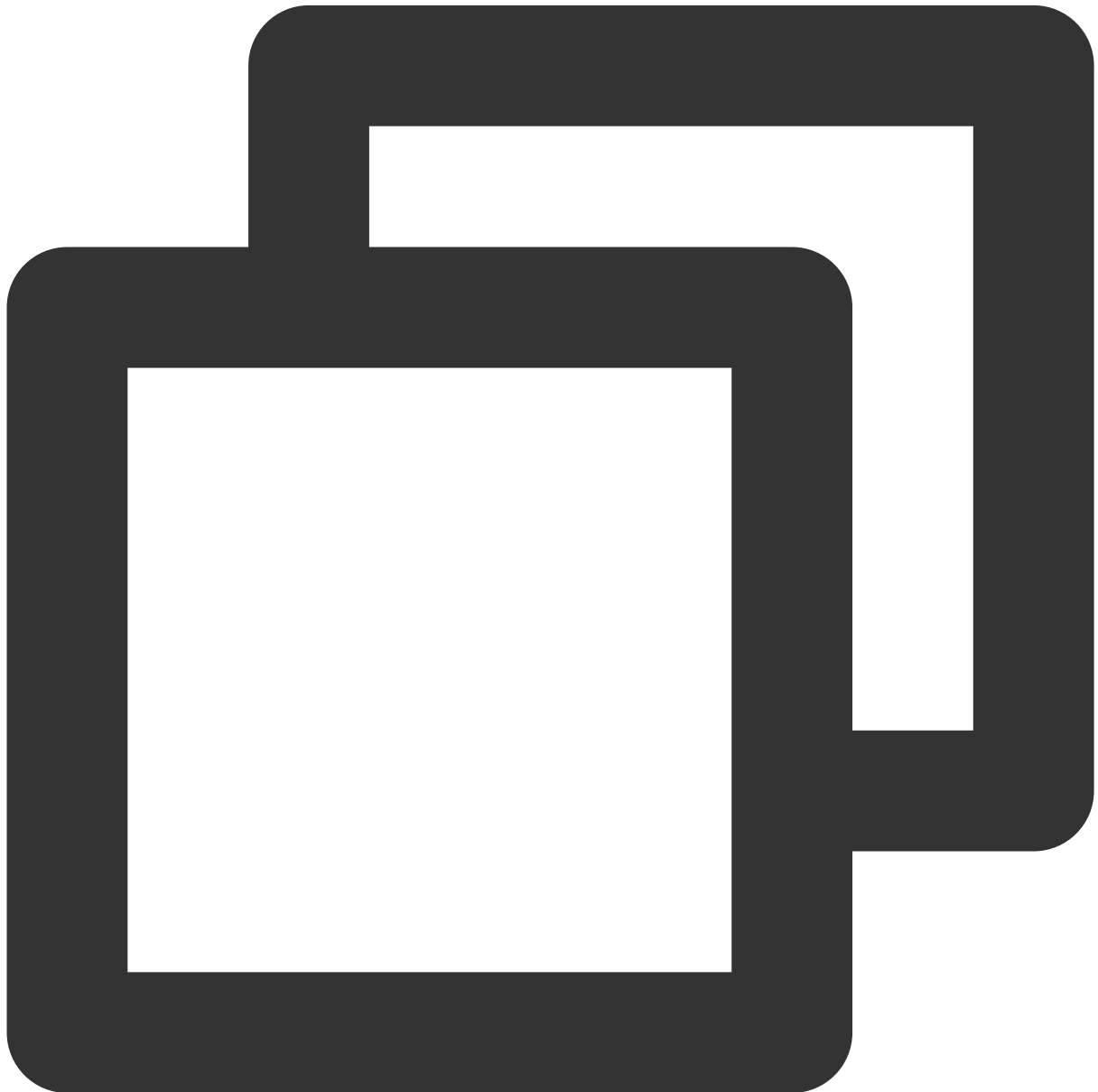
    TMASheetItemInfo *item2 = [[TMASheetItemInfo alloc] initWithTitle:@"click" type
        NSLog(@"click 点击");
    }];

    item2.icon = [UIImage imageNamed:@"icon_moreOperation_collect"];
```

```
[arrays addObject:item2];  
  
return arrays;  
}
```

2. 自定义弹出胶囊视图

通过 TMFMiniAppSDKDelegate 协议中的 `showShareViewWithTitle` 可以实现胶囊视图的自定义显示。

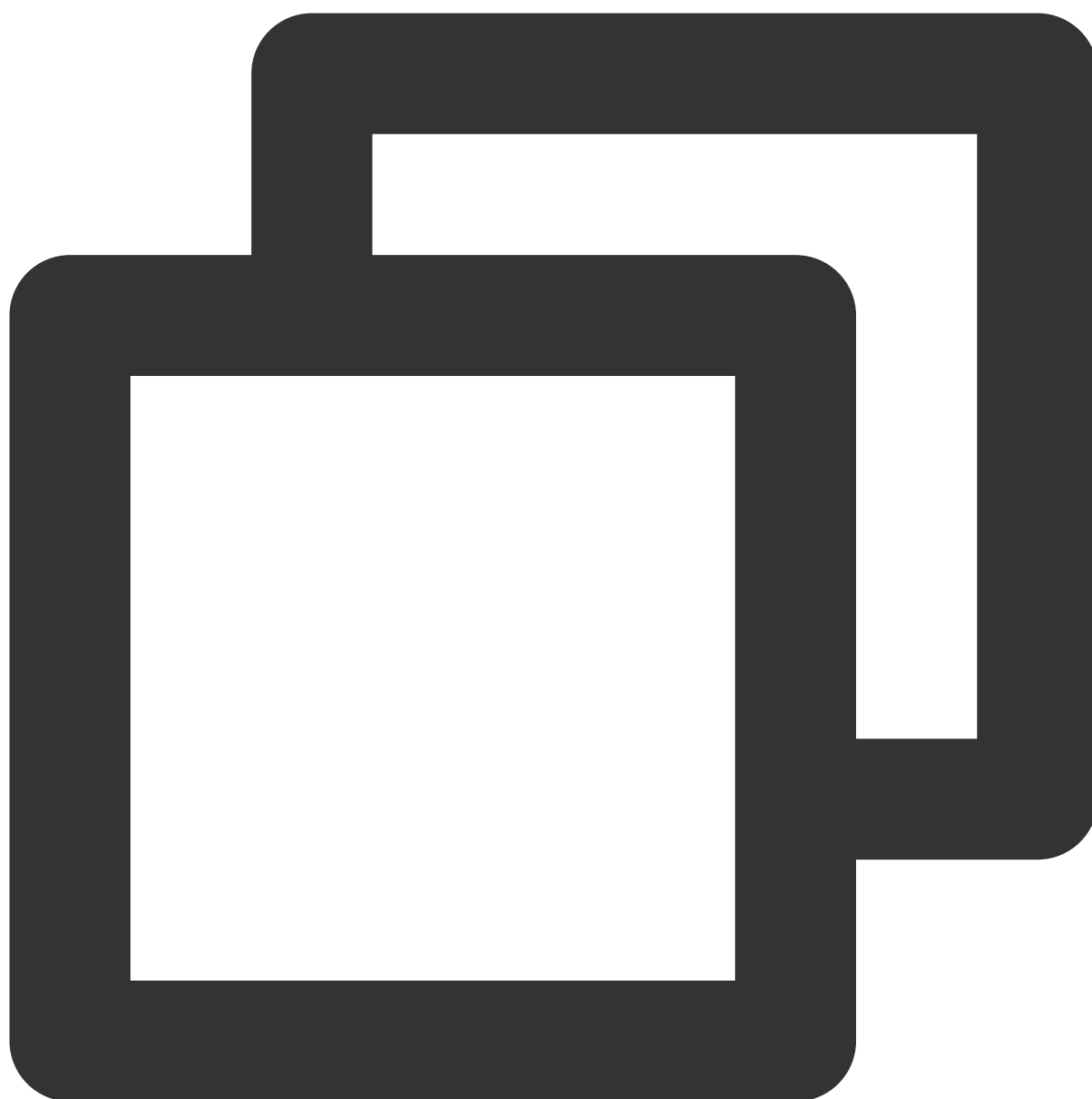


```
// 点击胶囊按钮呼起的面板 - Click the capsule button to call up the panel
```

```
// 如果此方法不实现，则会调用showActionSheetWithTitle: cancelButtonTitle:cancelAction:otl
// @param app 小程序信息 - Mini program information
// @param cancelButtonTitle 取消标题 - cancel title
// @param cancelAction 取消操作 - cancel the operation
// @param otherButtonTitleAndActions 其他按钮及响应操作 - other buttons and response o
// @param dismissBlock 面板收起后需要执行的操作（一定要调用以保证功能正确!!!） - The operat
// @param parentVC 呼起面板的vc - calls up the vc of the panel
- (void) showMoreButtonActionSheetWithApp: (TMFMiniAppInfo *) app
    cancelButtonTitle: (nullable NSString *) cancelButtonTitle
    cancelAction: (nullable dispatch_block_t) cancelAction
    otherButtonTitleAndActions: (nullable NSArray *) otherButtonTitleAndAct
    dismissBlock: (nullable dispatch_block_t) dismissBlock
    parentVC: (UIViewController *) parentVC;
```

设置小程序转场动画

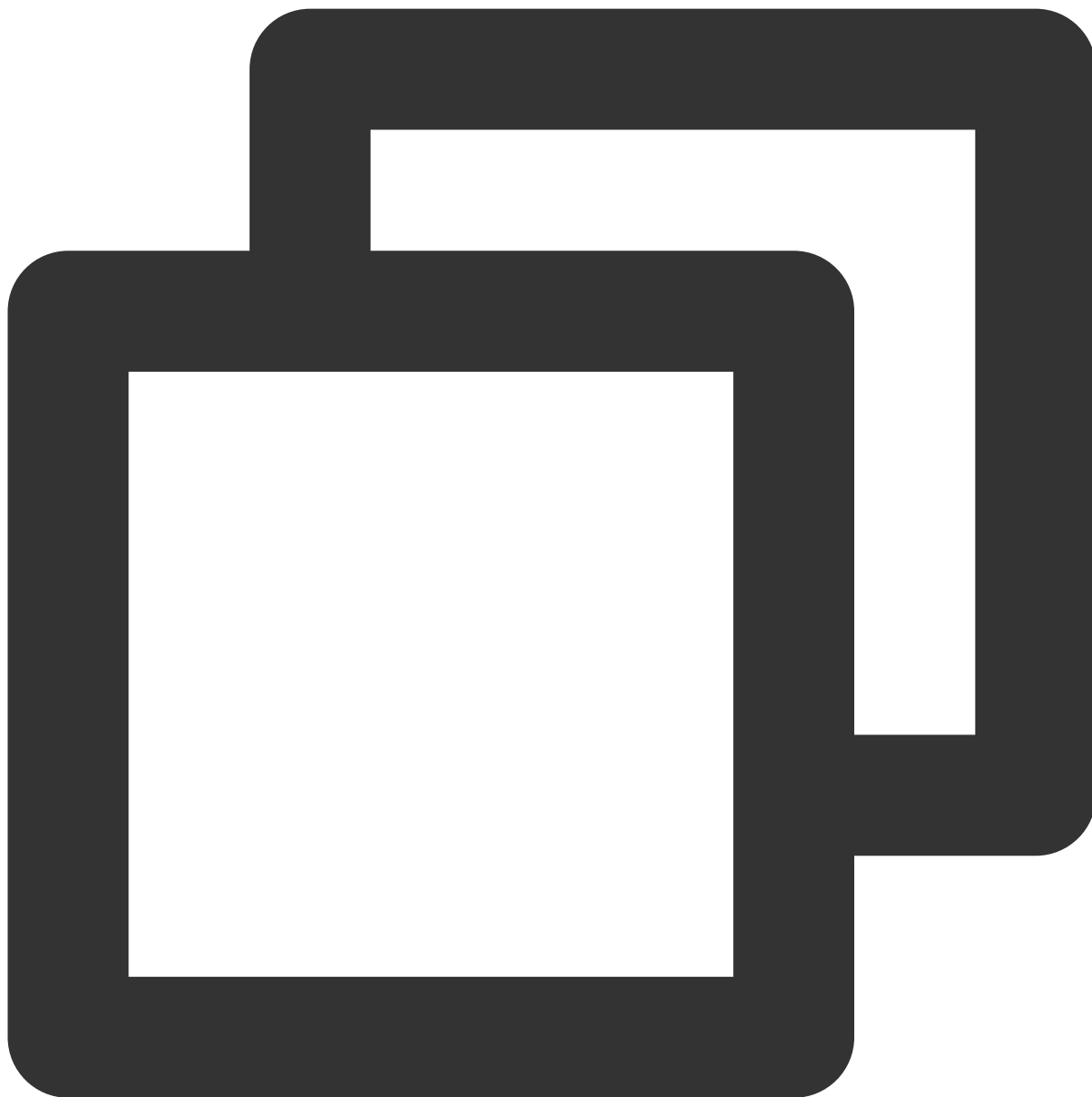
通过 `TMFMiniAppSDKDelegate` 协议中的 `getTMFSlideAnimationType` 可以实现小程序启动时的转场动画，目前支持下进上出、上进下出、左进右出、右进左出和默认类型（下进下出）。



```
// 设置小程序启动时转场动画为下进上出
- (TMFSlideAnimationType)getTMFSlideAnimationType{
    return TMFSlideAnimationTypeBottomToTop;
}
```

设置小程序权限弹框

通过 `TMFMiniAppSDKDelegate` 协议中的 `createAuthorizeAlertViewWithFrame` 可以实现小程序授权窗口的自定义，参数中包括小程序及需要申请的权限，宿主可以按照自己的风格实现对应的 `view` 并返回。

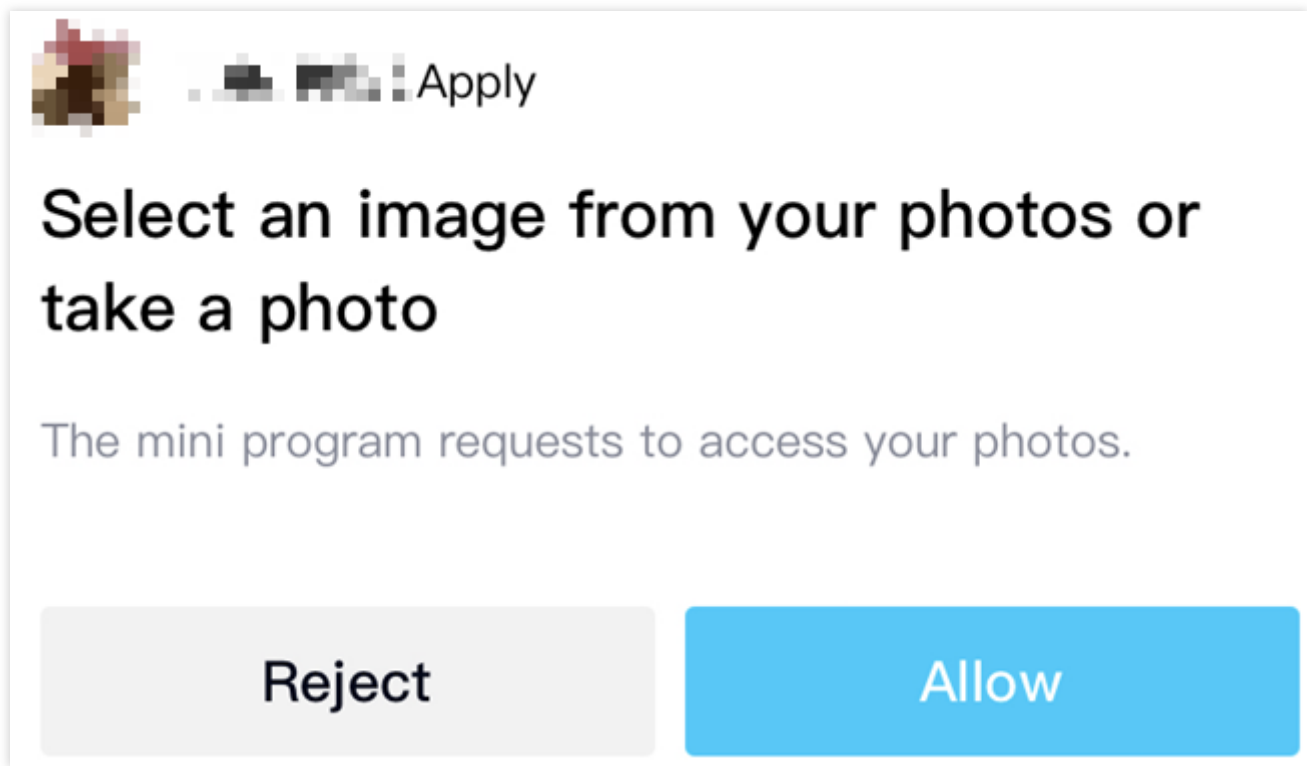


```
/**
 * @brief 创建自定义的授权窗口
 *
 * @param frame 窗口大小
 * @param scope 参考微信授权 scope
 * @param title 权限名称
 * @param desc 权限描述信息
 * @param privacyApi 当前调用的api
```



```
* @param appInfo 当前小程序信息
* @param allowBlock 允许回调
* @param denyBlock 拒绝回调
*/

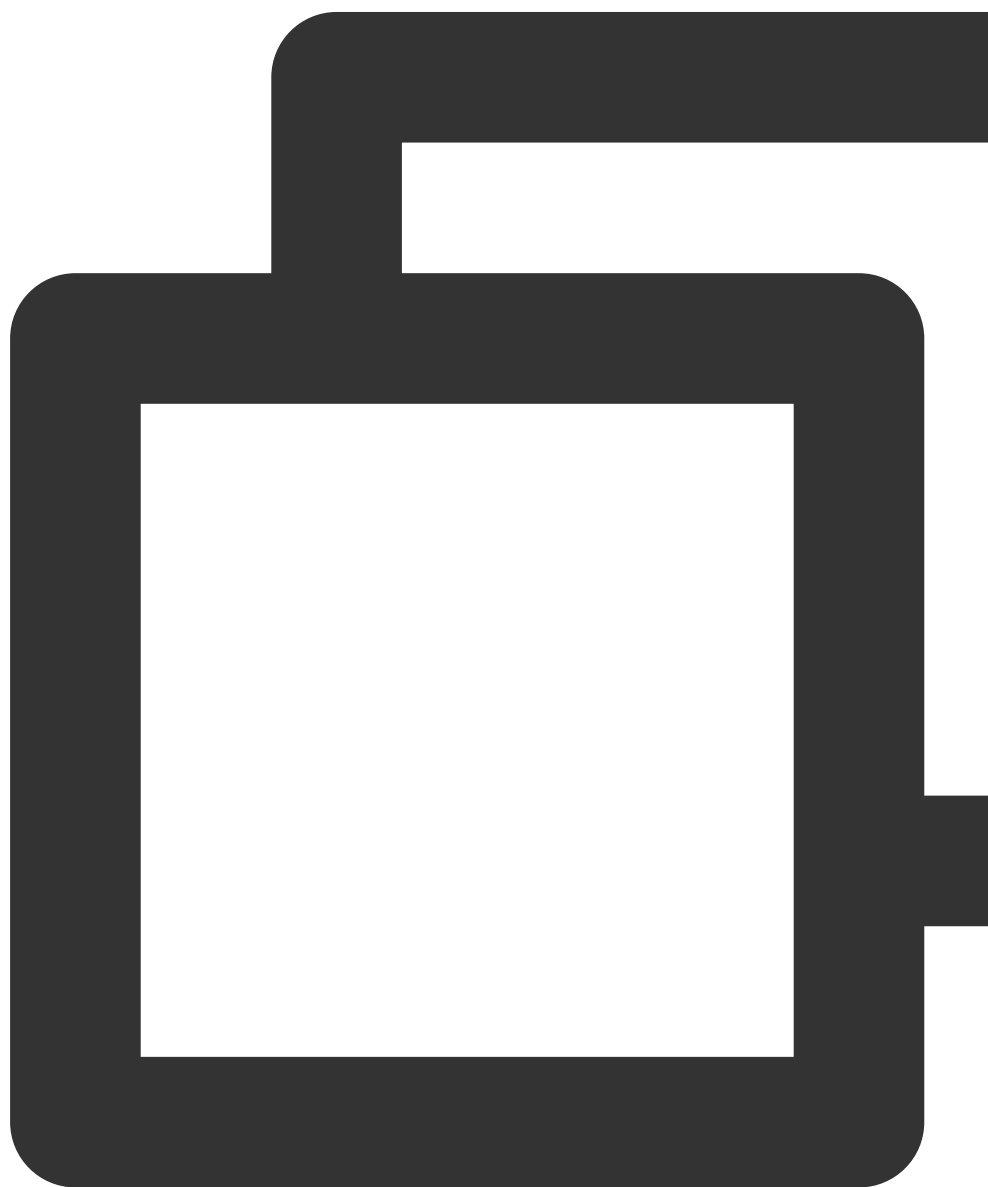
- (UIView *)createAuthorizeAlertViewWithFrame:(CGRect) frame
    scope:(NSString *)scope
    title:(NSString *)title
    desc:(NSString *)desc
    privacyApi:(NSString *)privacyApi
    appInfo:(TMFMiniAppInfo *_Nullable) appInfo
    allowBlock:(void (^)(void))allowBlock
    denyBlock:(void (^)(void))denyBlock;
```



设置小程序内部 UI

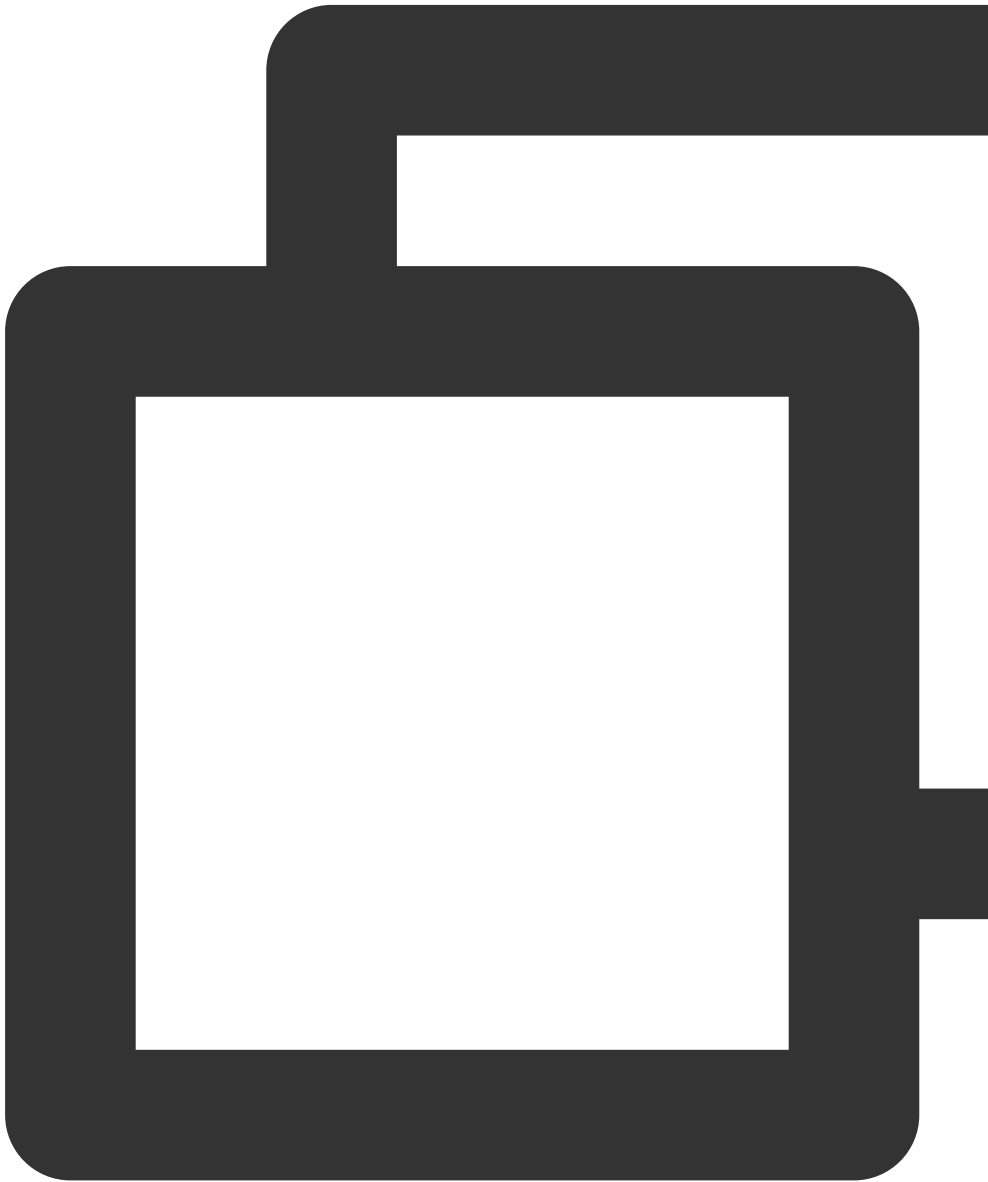
TCMPP 也支持小程序内部使用的 ui 自定义显示，在 `TMFMiniAppSDKDelegate` 实现对应的方法即可，目前支持的内容如下：

小程序 api	TMFMiniAppSDKDelegate 方法
<code>wx.showLoading</code>	



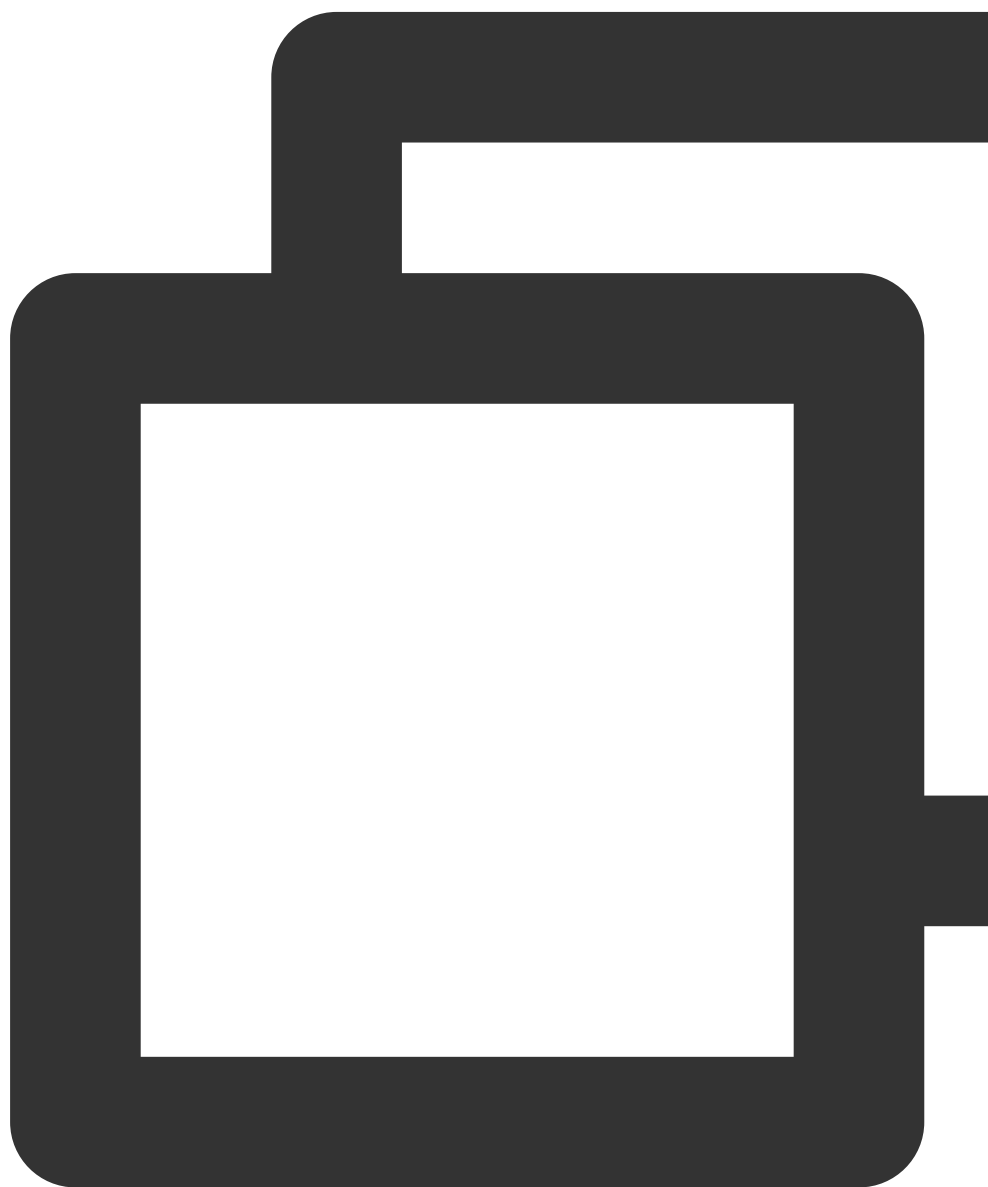
```
- (void)showLoading:(TMALoadingInfo * _Nullable)infos;
```

wx.hideLoading



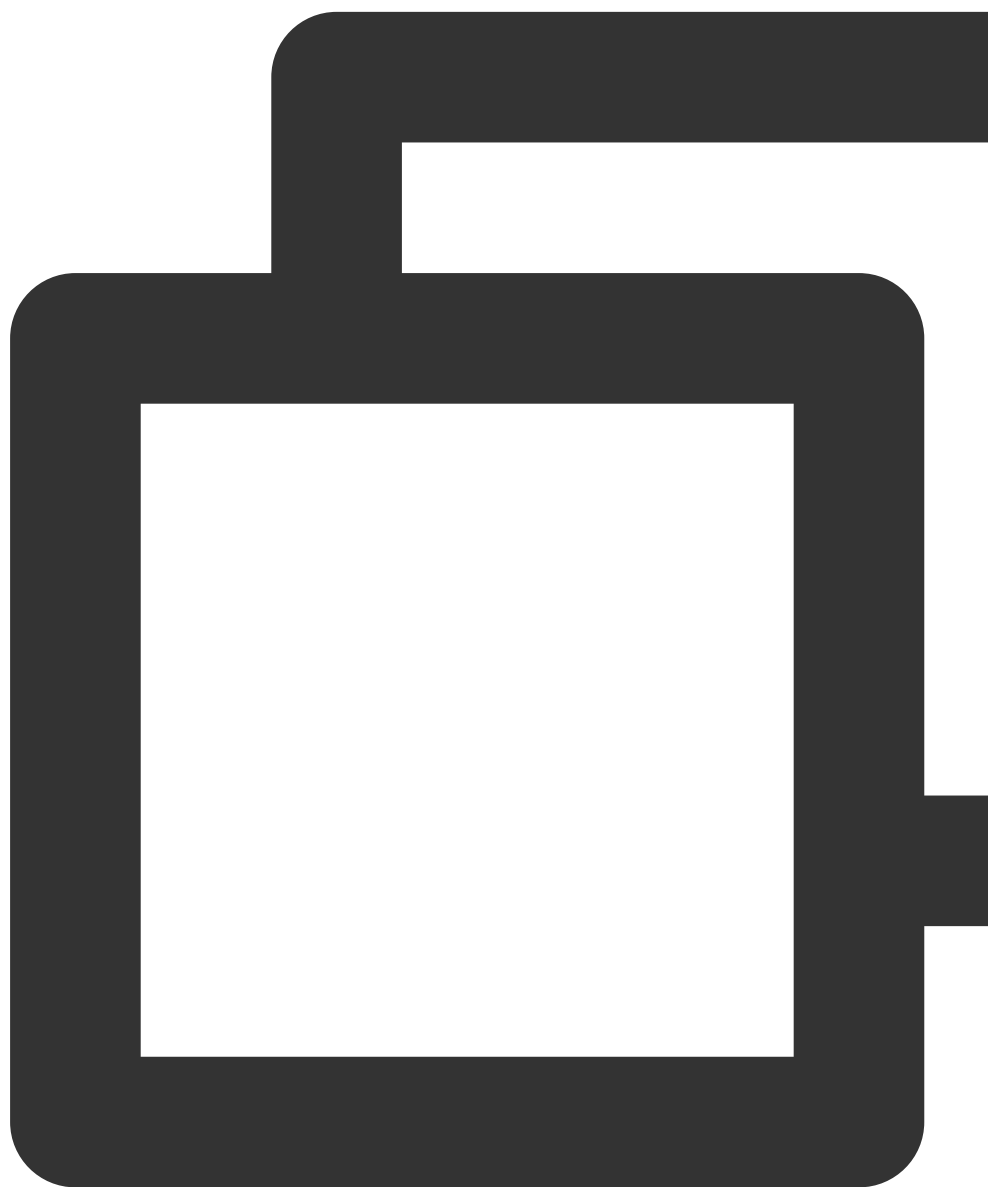
```
- (void)hideLoading;
```

wx.showToast



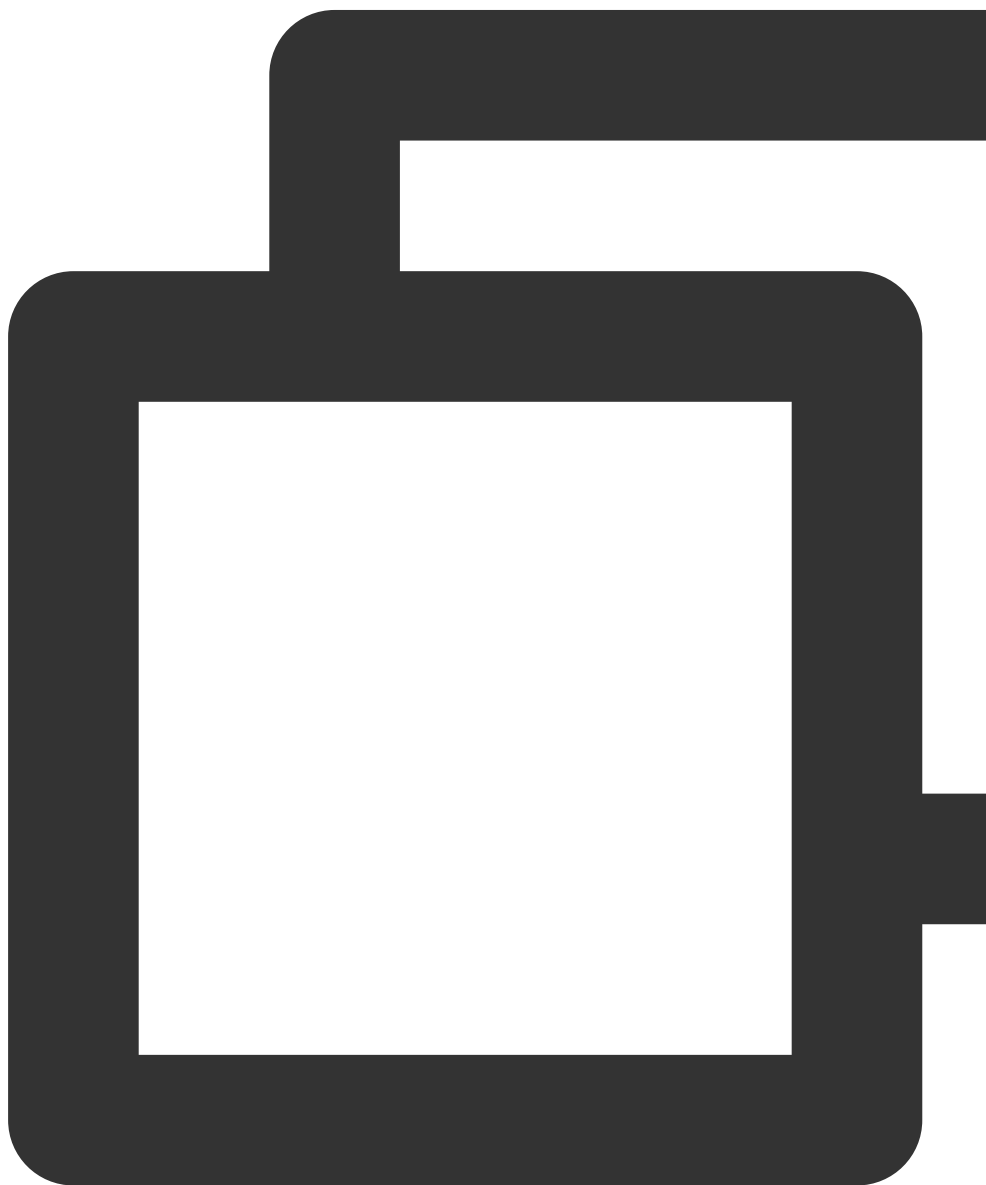
```
- (void) showToast:(TMAToastInfo *)infos;
```

wx.hideToast



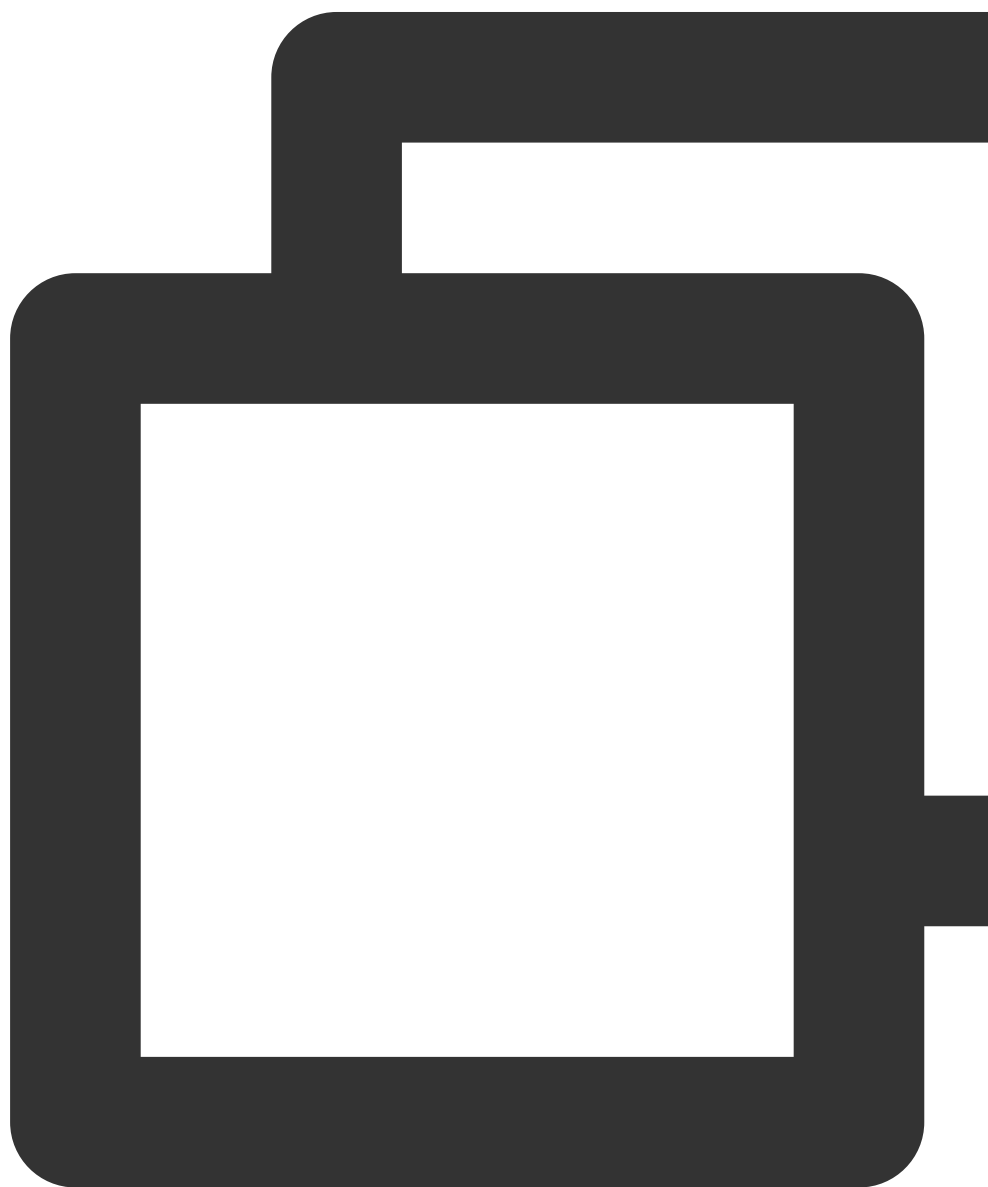
```
- (void)hideToast;
```

`wx.showActionSheet`
(`actionSheetType = 0`)



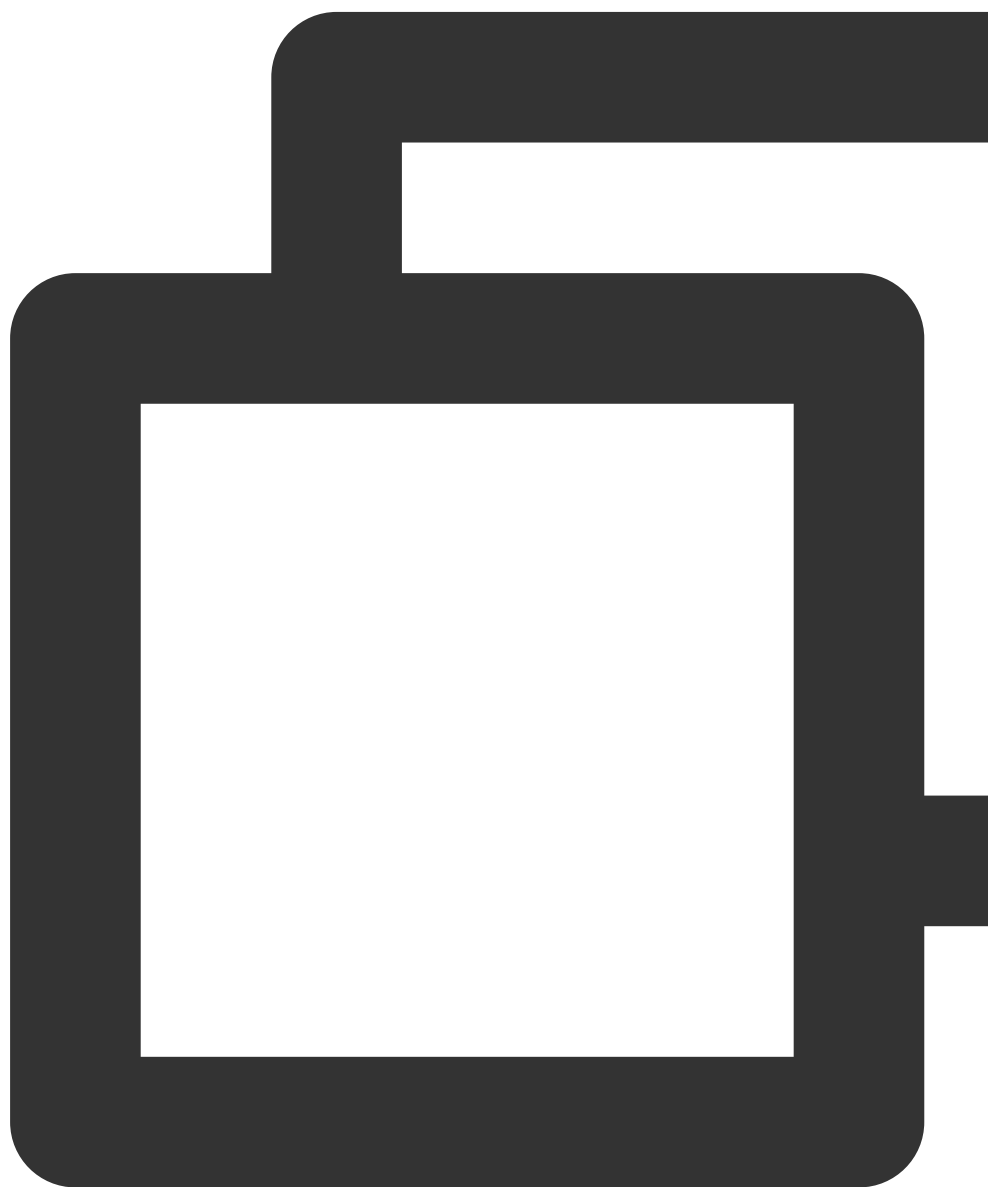
```
- (void)showActionSheetWithTitle:(nullable NSString *)title
    cancelButtonTitle:(nullable NSString *)cancelBut
    cancelAction:(nullable dispatch_block_t)can
    otherButtonTitleAndActions:(nullable NSArray *)otherButto
    dismissBlock:(nullable dispatch_block_t)dis
    presentingViewController:(UIViewController *)presenting
```

wx.showActionSheet
(actionSheetType =
1)



```
- (void) showShareViewWithTitle:(nullable NSString *)title
    cancelAction:(nullable dispatch_block_t)cancelAction
    otherButtonTitleAndActions:(nullable NSArray *)otherButtonTitlesAndActions
    dismissBlock:(nullable dispatch_block_t)dismissBlock
    parentVC:(UIViewController *)parentVC;
```

wx.showModal



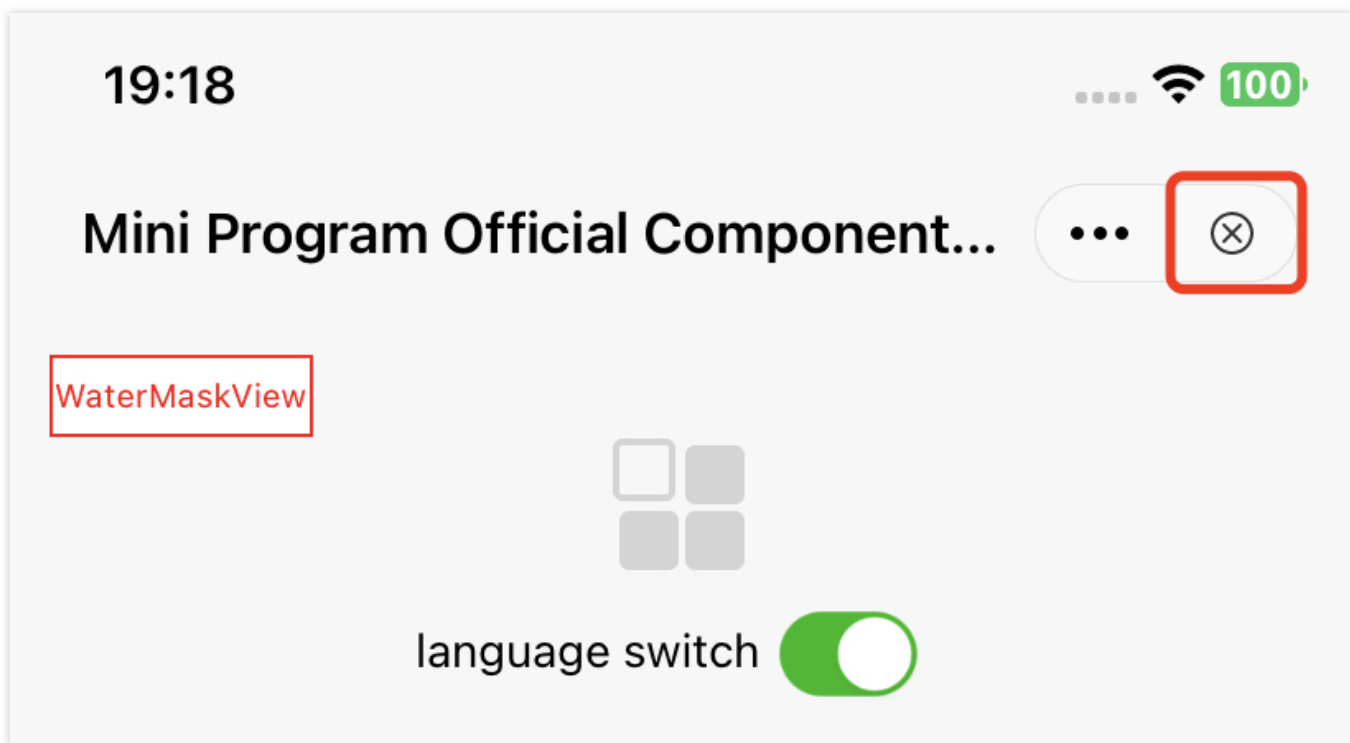
```
- (void) showAlertWithTitle:(nullable NSString *)title  
    withMessage:(nullable NSString *)message  
    actionBlocks:(nullable NSArray<AlertActionInfo*> *)actionBlocks  
    presentingViewController:(UIViewController*)presentingV
```

定制胶囊按钮功能

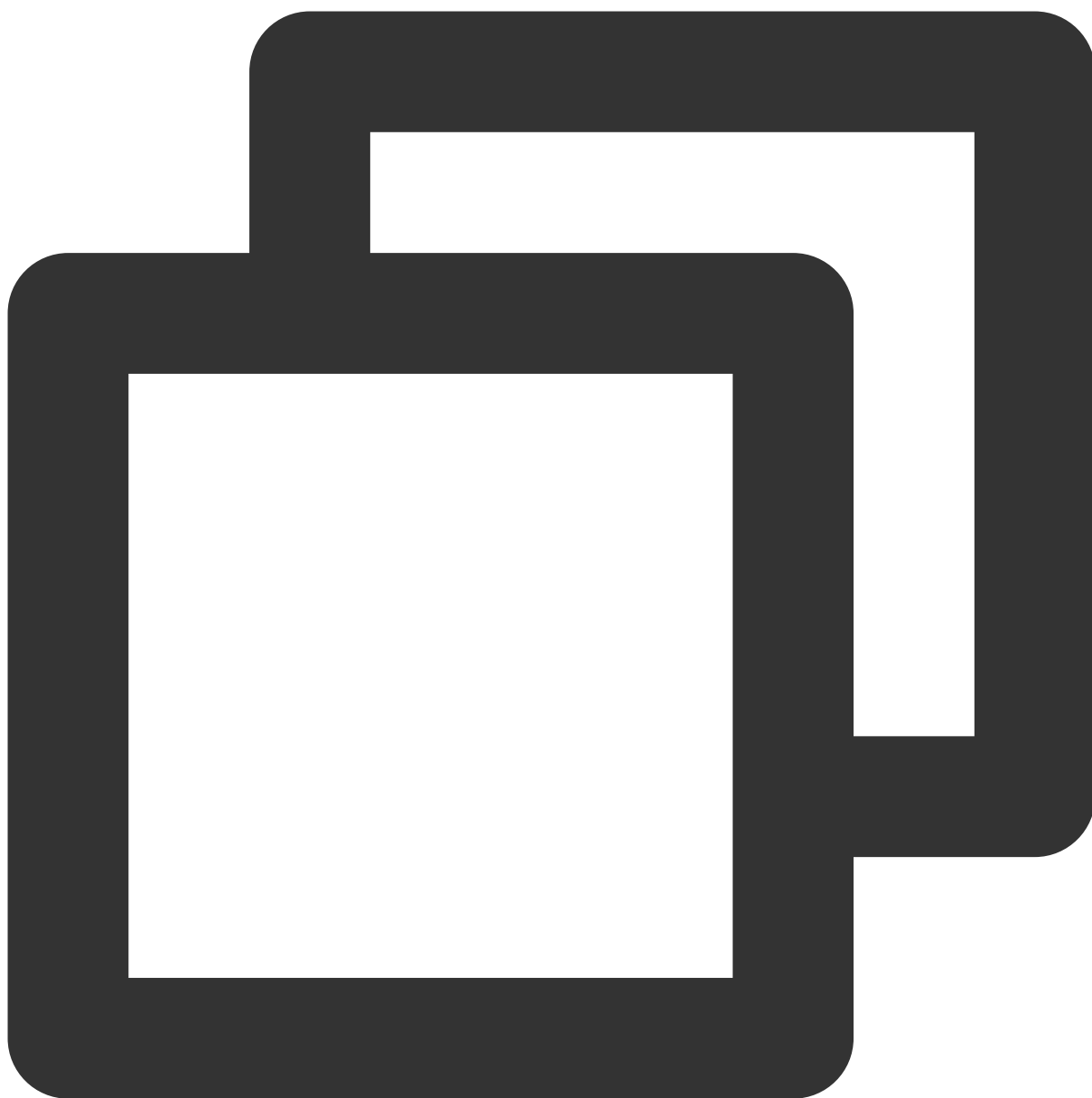
定制关闭按钮事件监听

定制胶囊按钮的关闭事件监听，能够让宿主应用在关闭按钮点击时获取到对应的回调事件。

关闭按钮示意图：



API 说明：



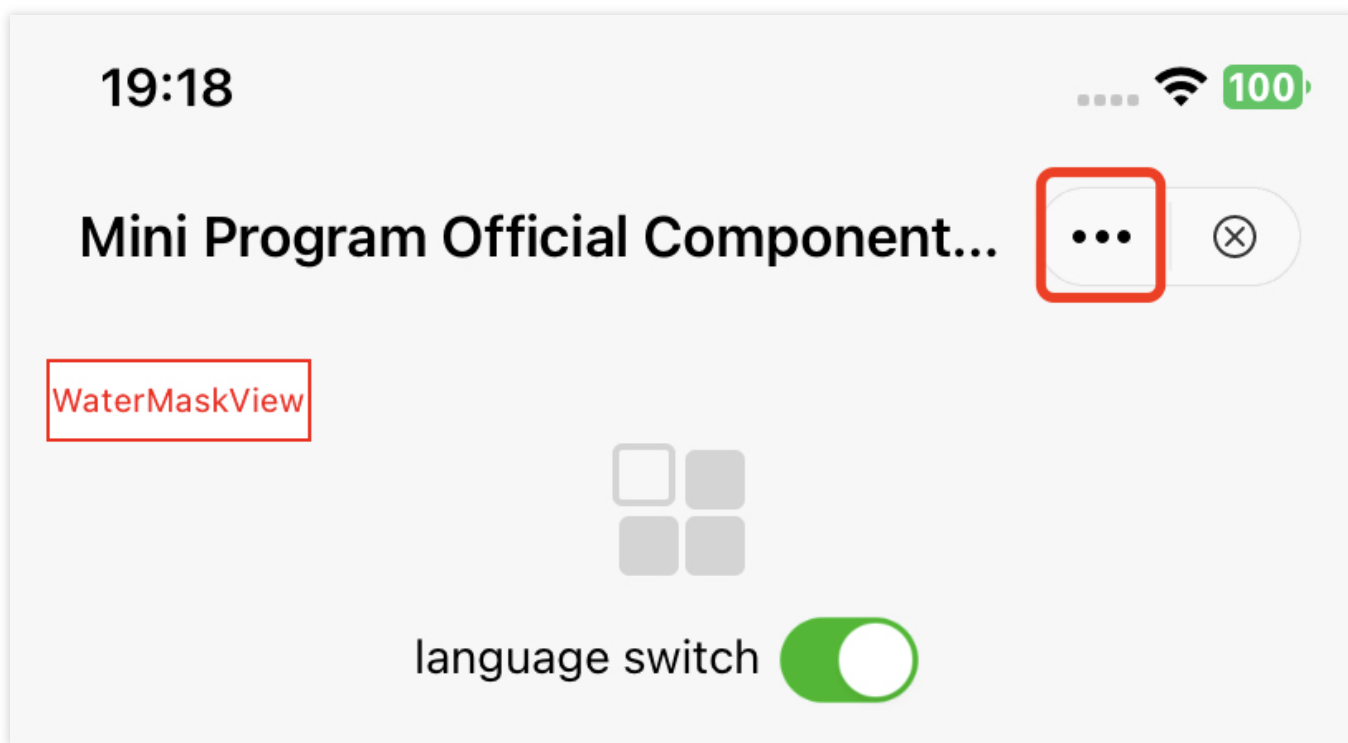
```
#pragma mark - 退出挽留 - Exit retention
- (BOOL)shouldDetainUser:(TMFMiniAppInfo *)app;
```

单击关闭时，会触发此 API，如果返回 YES，则会出现弹窗挽留用户，如果返回 NO，则会直接退出小程序。

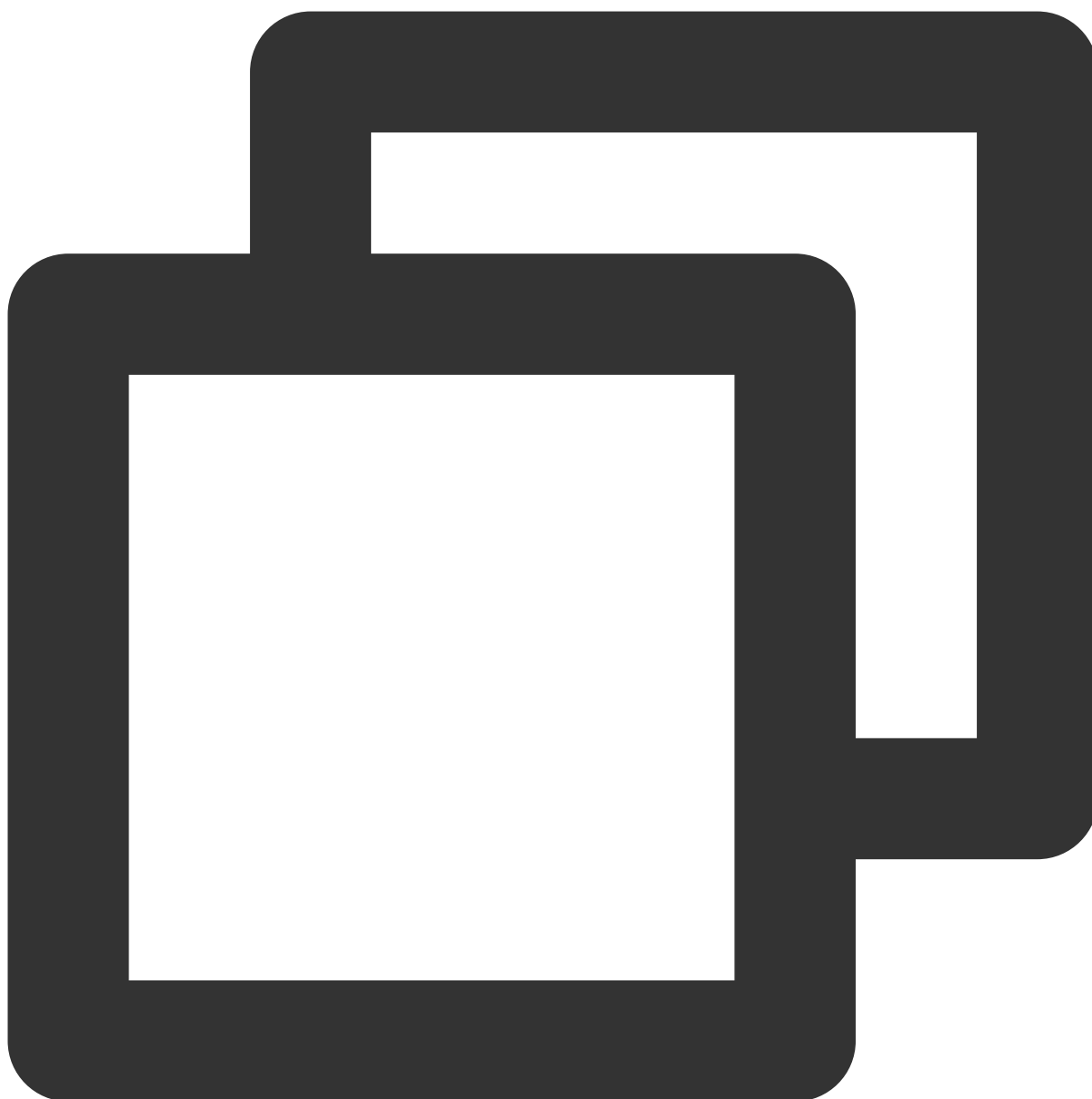
定制更多按钮事件监听

定制更多按钮的事件监听，能够让宿主应用在单击更多按钮单击时监听到到对应 item 的回调事件。

更多按钮示意图：



API 说明：



```
// 点击胶囊按钮呼起的面板 - Click the capsule button to call up the panel
// 如果此方法不实现，则会调用showActionSheetWithTitle: cancelButtonTitle:cancelAction:otl
// @param app 小程序信息 - Mini program information
// @param cancelButtonTitle 取消标题 - cancel title
// @param cancelAction 取消操作 - cancel the operation
// @param otherButtonTitleAndActions 其他按钮及响应操作 - other buttons and response o
// @param dismissBlock 面板收起后需要执行的操作（一定要调用以保证功能正确!!!） - The operat
// @param parentVC 呼起面板的vc - calls up the vc of the panel

- (void)showMoreButtonActionSheetWithApp:(TMFMiniAppInfo *)app
```

```
cancelButtonTitle:(nullable NSString *)cancelButtonTitle
cancelAction:(nullable dispatch_block_t)cancelActio
otherButtonTitleAndActions:(nullable NSArray *)otherButtonTitleAn
dismissBlock:(nullable dispatch_block_t)dismissBloc
parentVC:(UIViewController *)parentVC;
```

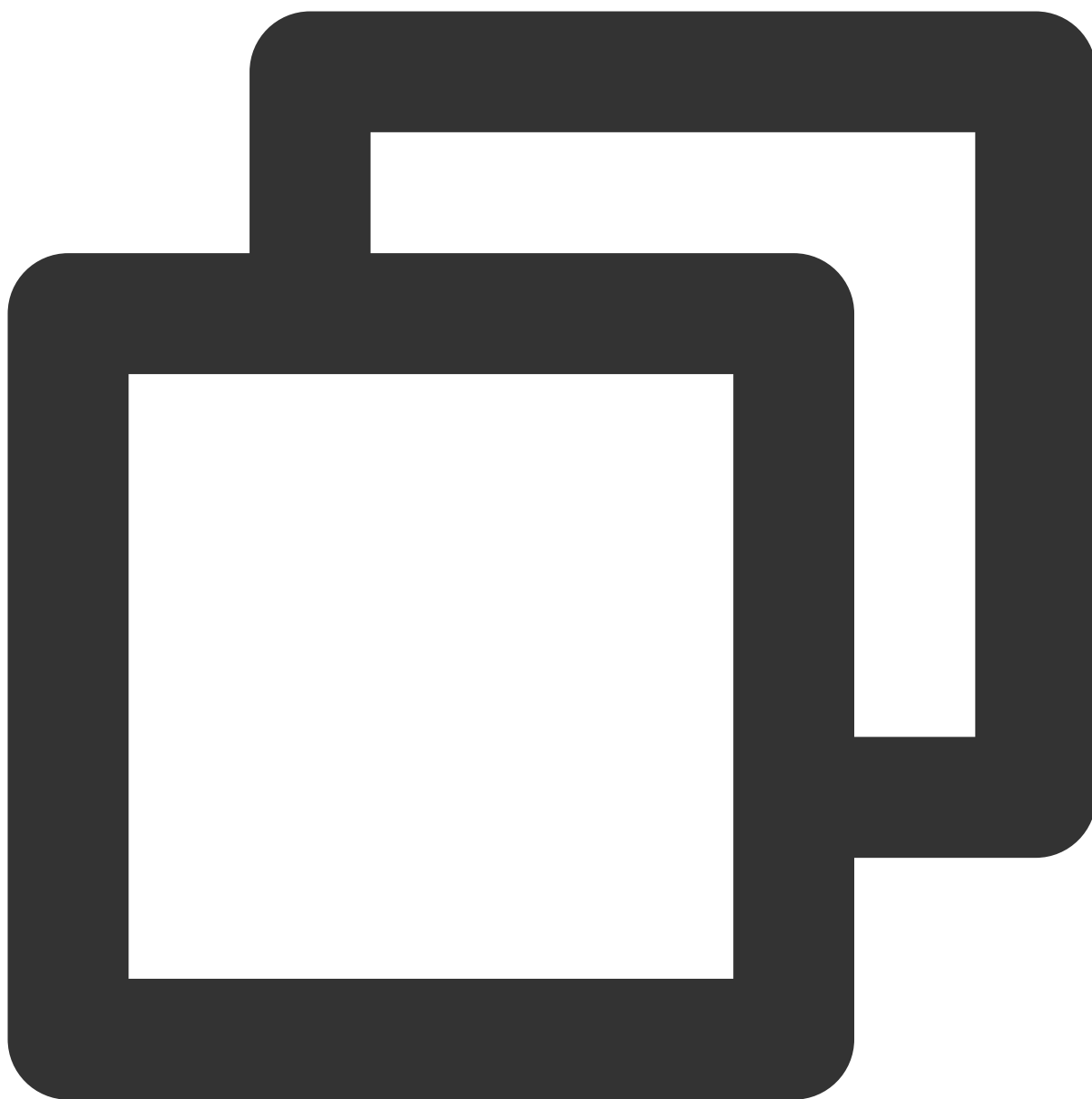
定制更多按钮展示列表

到用户触发更多按钮点击事件时，会弹出如下的可选扩展按钮列表，默认列表示意图如下：



通过重写 `customizedConfigForShare` 方法，可以自定义分享途径、决定展示顺序。

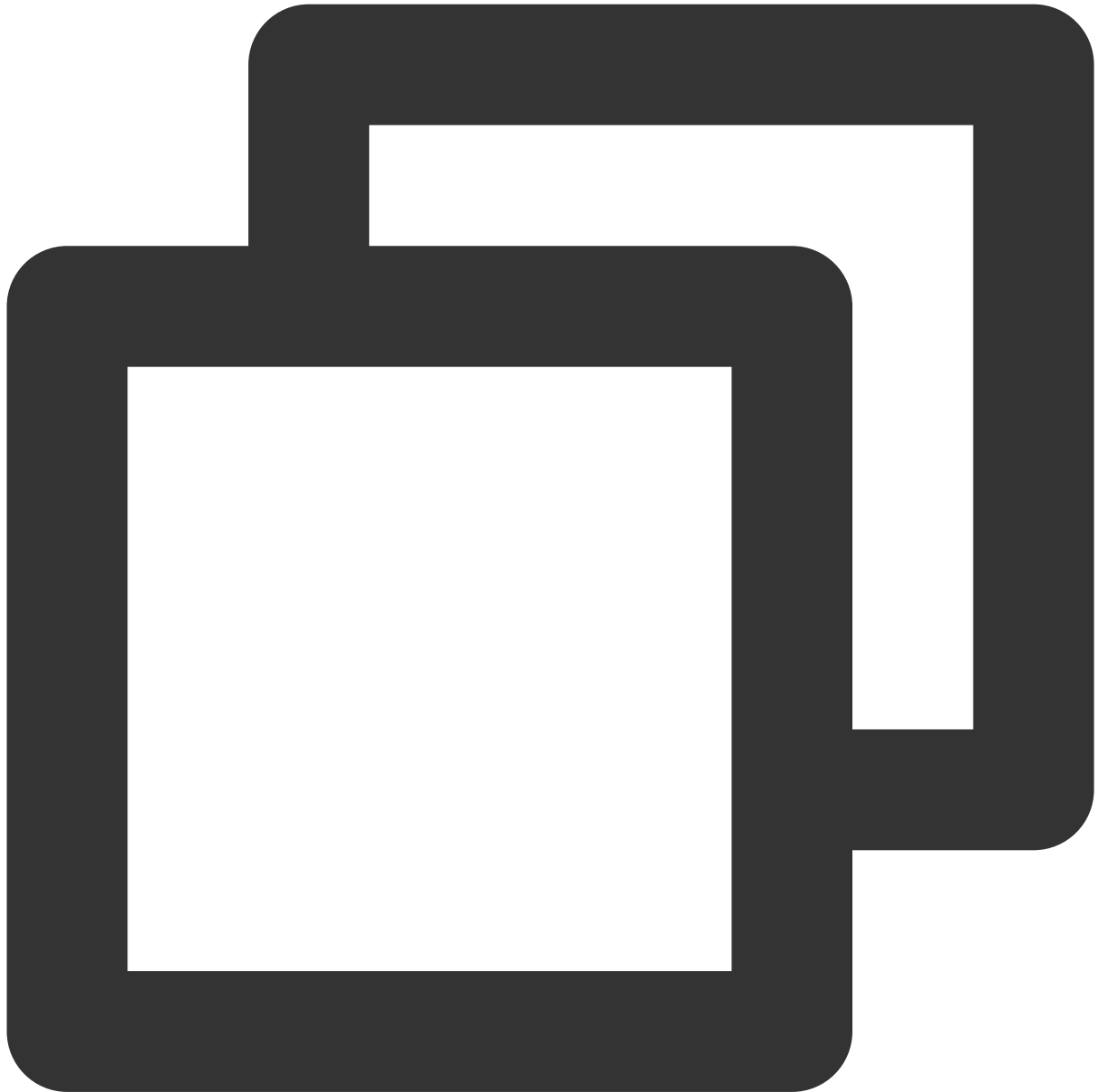
API 说明：



```
// 宿主App可以自定义分享途径、决定展示顺序，目前使用在点击更多按钮、button组件(open-type="share")
// 1、默认渠道：QQ好友、QQ空间、微信、朋友圈（具体type参见MAUIDelegateShareViewType），由开发
// 2、自定义分享渠道：宿主App自定义（type填MAUIDelegateShareViewTypeCustomizedShare，自定义
// 3、自定义事件处理：宿主App自定义（type填MAUIDelegateShareViewTypeCustomizedAction）
// 以上三种渠道展示顺序支持混排
///
// The host App can customize the sharing path and determine the display order. It
// 1. Default channels: QQ Friends, QQ Space, WeChat, Moments (for specific types,
// 2. Customized sharing channel: Host App customization (type fills in MAUIDelegateShareViewTypeCustomizedShare)
// 3. Custom event processing: Host App customization (type fills in MAUIDelegateShareViewTypeCustomizedAction)
```

```
// The display order of the above three channels  
- (NSArray<TMASheetItemInfo *> *)customizedConfigForShare;
```

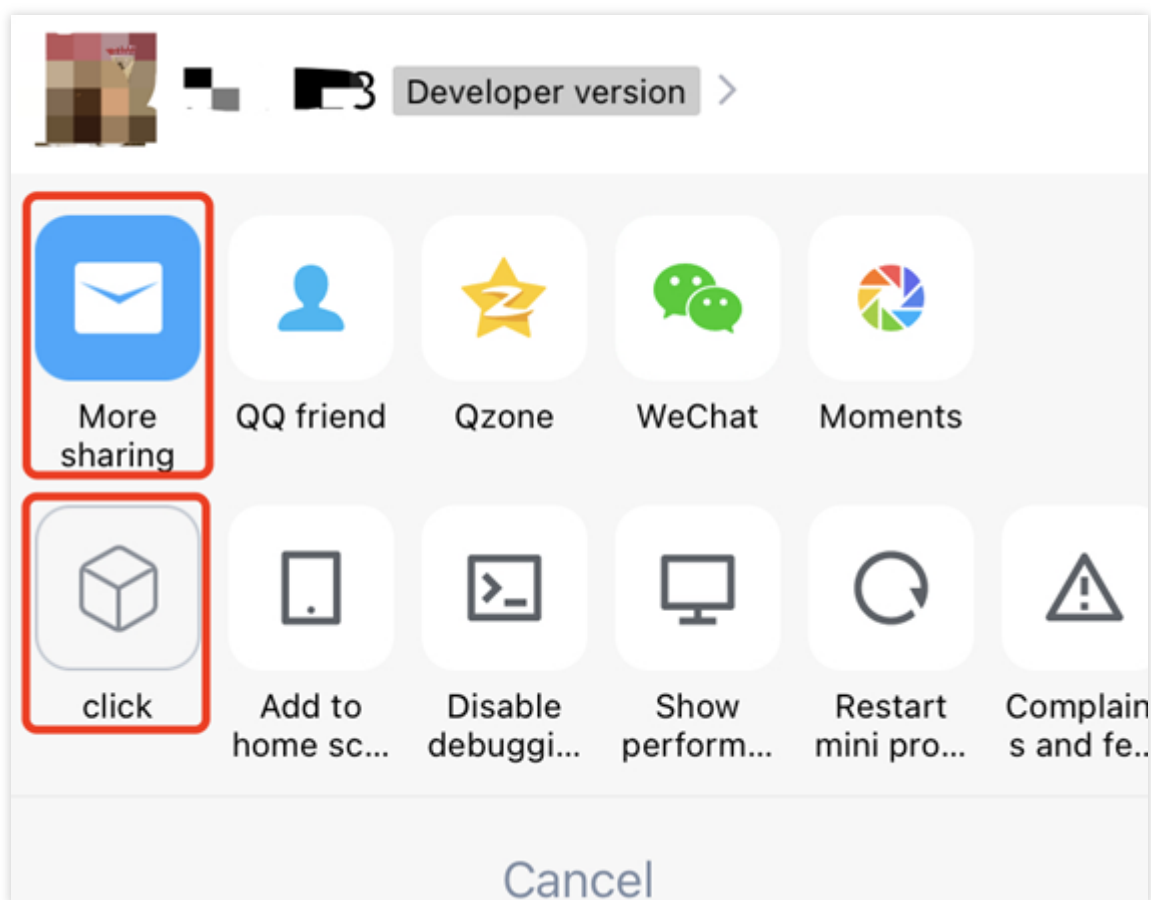
示例代码：



```
- (NSArray<TMASheetItemInfo *> *)customizedConfigForShare {  
    NSMutableArray *arrays = [[NSMutableArray alloc] init];  
    TMASheetItemInfo *item1 = [[TMASheetItemInfo alloc] initWithTitle:@"More sharin  
    item1.icon = [UIImage imageNamed:@"icon_moreOperation_shareChat"];  
    [arrays addObject:item1];  
}
```

```
TMASheetItemInfo *item2 = [[TMASheetItemInfo alloc] initWithTitle:@"click" type
    NSLog(@"click 点击");
}];
item2.icon = [UIImage imageNamed:@"icon_moreOperation_collect"];
[arrays addObject:item2];
return arrays;
}
```

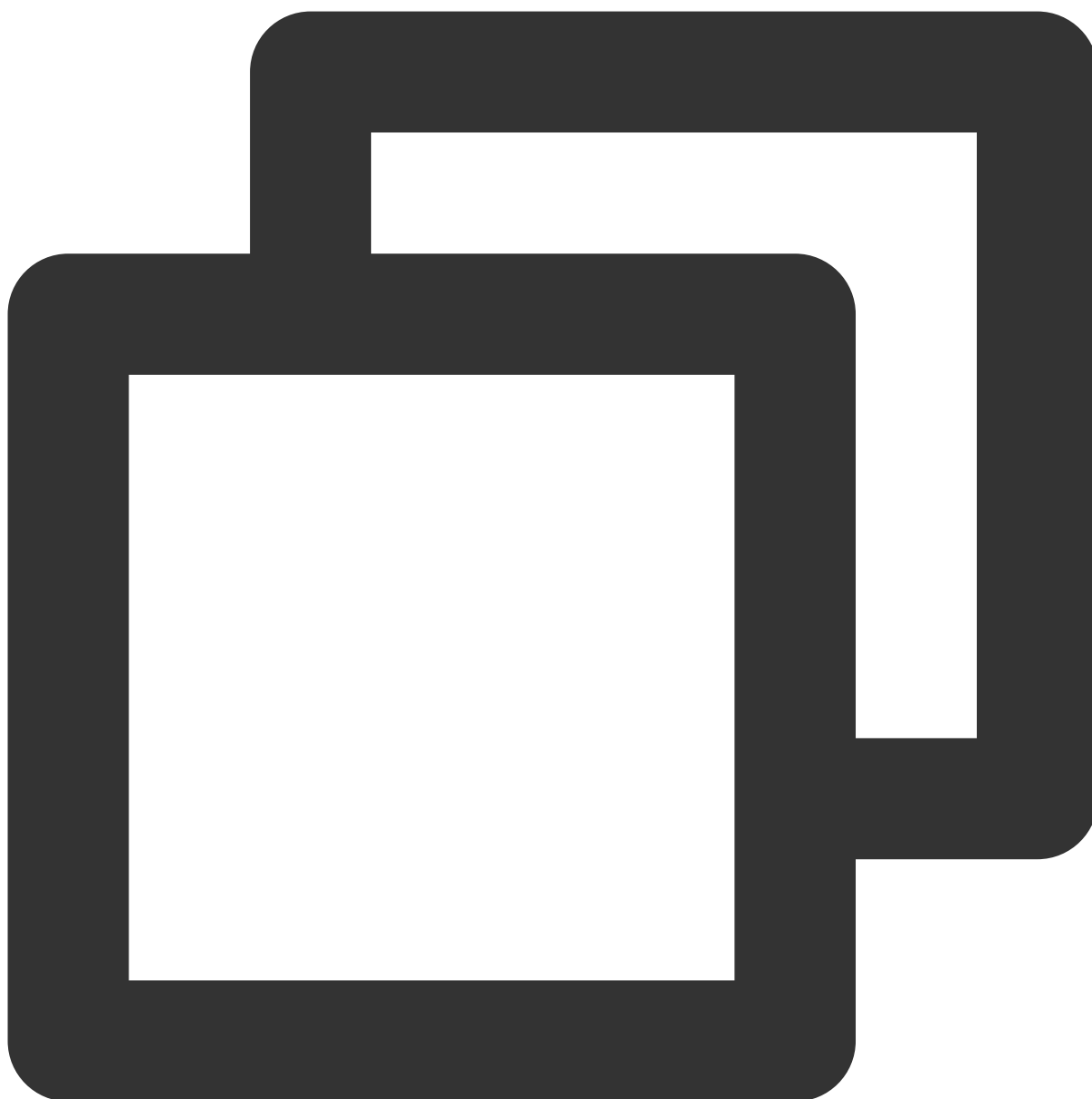
效果如下：



定制图片选择

小程序 SDK 提供默认的图片选择，默认使用 Android 系统自带的相册选择器进行选择，在小程序调用多媒体选择 (wx.chooseImage) 触发。

定制图片选择可以通过重写 selectMediaFromPickerWithModel 方法实现。



```
// 从选图器选择媒体 - Select media from the image picker
// @param model 配置 - configuration
// @param parentVC vc
// @param completionBlock 选择完毕后需要回传数据 根据所选类型接受 TMAPickerImageModel TMAI
- (void)selectMediaFromPickerWithModel:(TMAMediaChooseConfigModel *)model
    parentVC:(UIViewController *)parentVC
    completionBlock:(void (^)(NSArray * _Nullable medias, NSError * _Nullable error))completionBlock {
    // 拍摄媒体 - Shooting media
```

```
// @param model 配置 - configuration
// @param parentVC vc
// @param completionBlock 选择完毕后需要回传数据, 根据所选类型接受TMACameraImageModel TMAC
- (void)selectMediaFromCameraWithModel:(TMAMediaChooseConfigModel *)model
    parentVC:(UIViewController *)parentVC
    completionBlock:(void(^)(id _Nullable media, NSError * _Null

// 从PHAsset中获取图片数据 - Get image data from PHAsset
// @param phAsset 媒体对象 - media object
// @param needCompress 是否需要压缩 - whether compression is required
- (NSData *)imageDataFromPhAsset:(PHAsset *)phAsset needCompress:(BOOL)needCompress
```

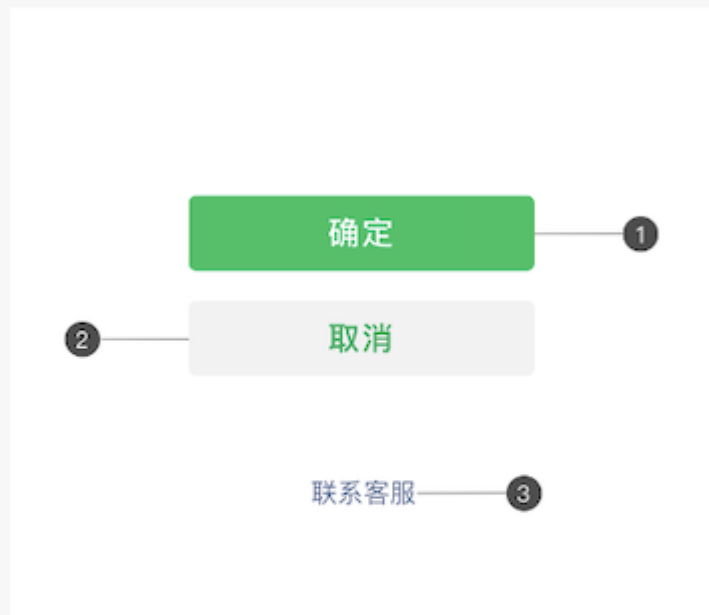
定制图片预览

小程序 SDK 提供默认的图片预览实现, 在小程序调用多媒体选择(`wx.previewImage`)时触发。

定制图片预览可以通过重写- `(void)navigationController:(UINavigationController *)navigationController
presentImageWithCurrentUrl:(NSString *)currentAbsoluteUrl imageUrlArray:(NSArray *)absUrlsInPreviewArray`方法实现。

默认图片预览页面如下图：

多按钮示意



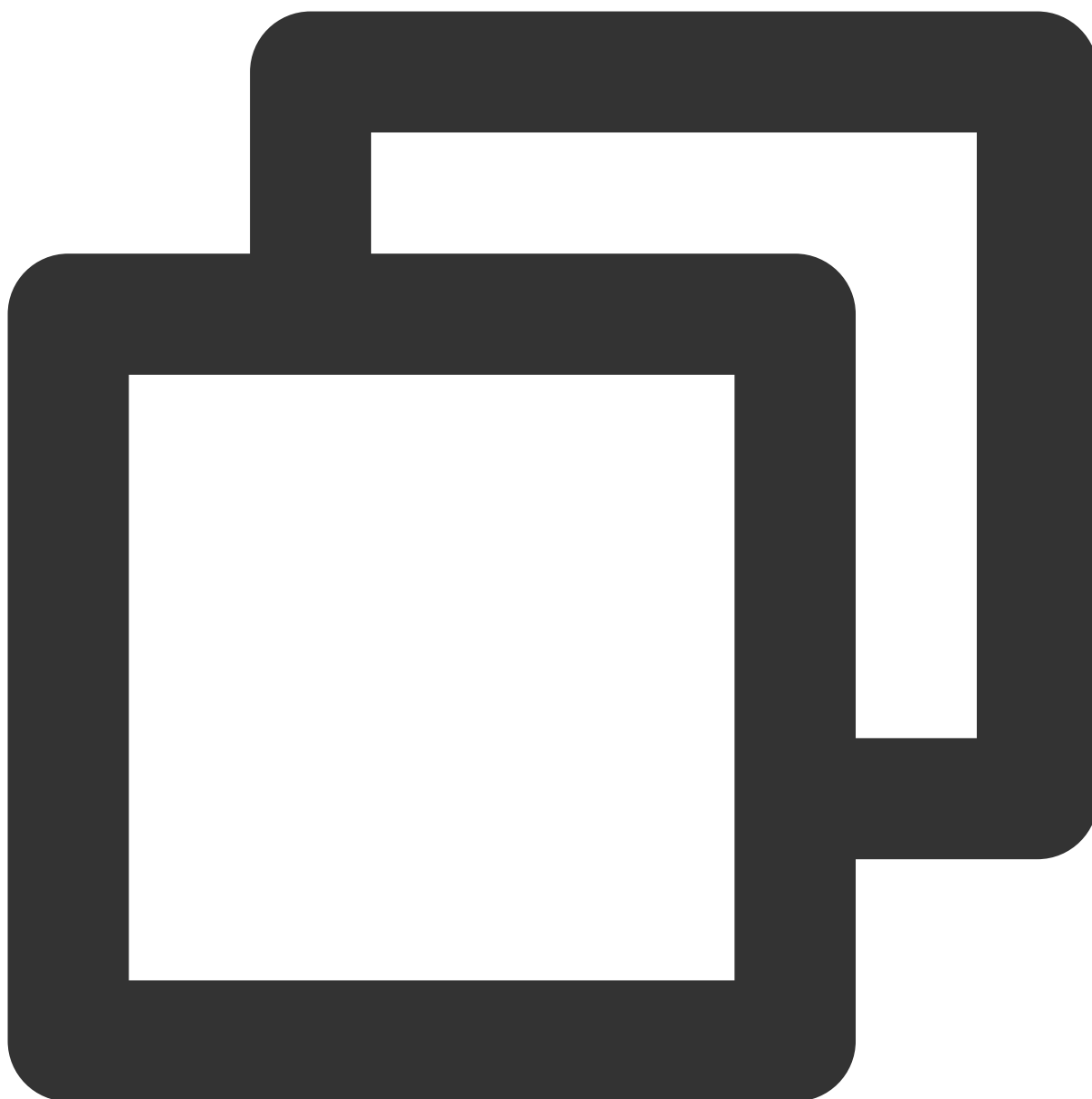
请这样设计

并列多个操作时，需要有明确的主次之分：

1. 主要操作
2. 辅助操作
3. 文字链操作

请勿这样设计

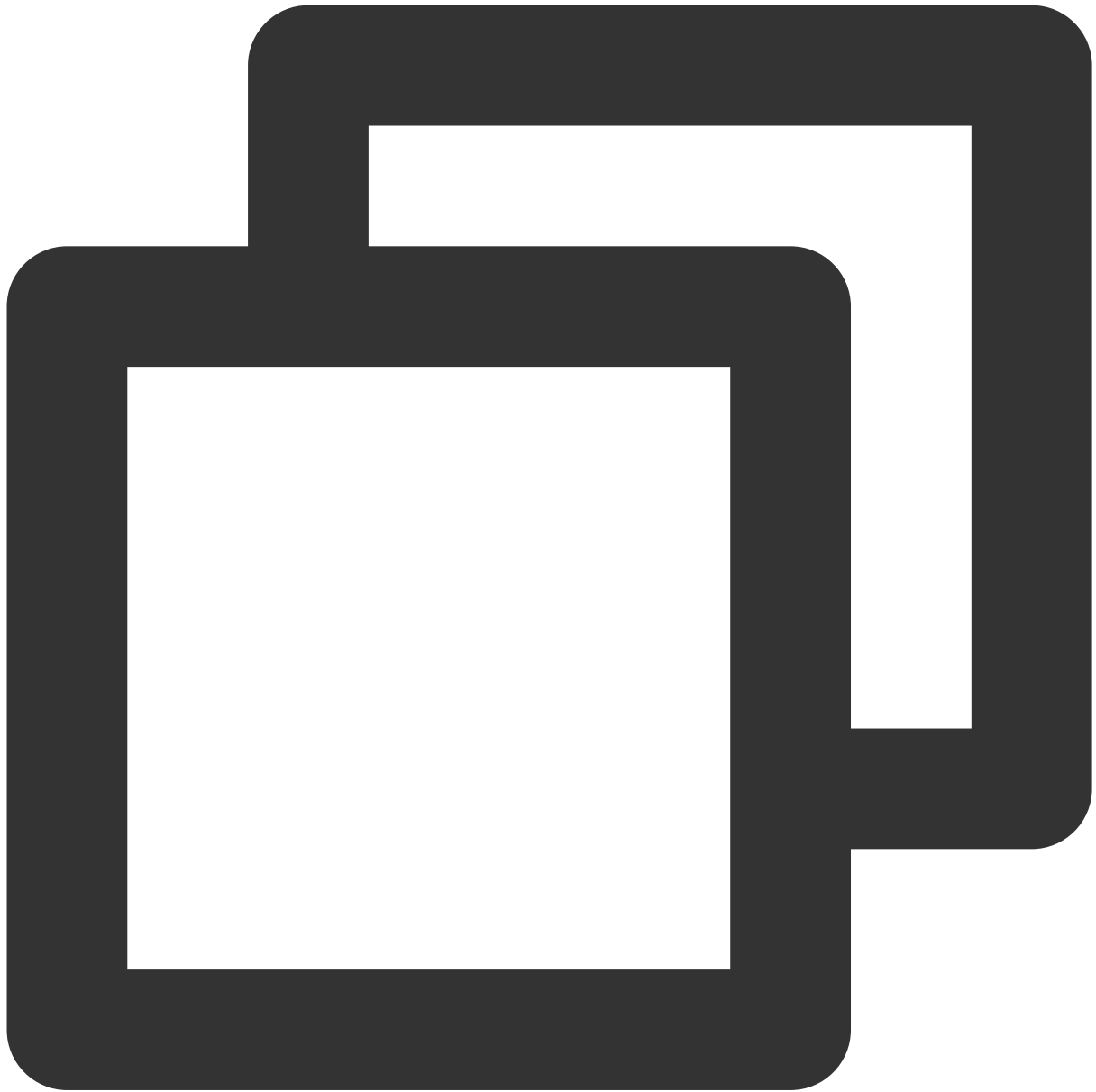
多个平行操作，让用户无从选择。



```
// 图片预览 - Image Preview
// @param navigationController 呼起图片预览的导航栏 - calls up the navigation bar for :
// @param currentAbsolutePath 当前页面地址 - current page address
// @param absUrlsInPreviewArray 需要预览的图片 - Pictures to be previewed
- (void)navigationController:(UINavigationController *)navigationController
  presentImageWithCurrentUrl:(NSString *)currentAbsolutePath
        imageUrlArray:(NSArray *)absUrlsInPreviewArray;
```

定制文档预览

iOS SDK 提供了默认的文档预览功能，如果你想自己实现文档预览功能，请重写下面的接口：



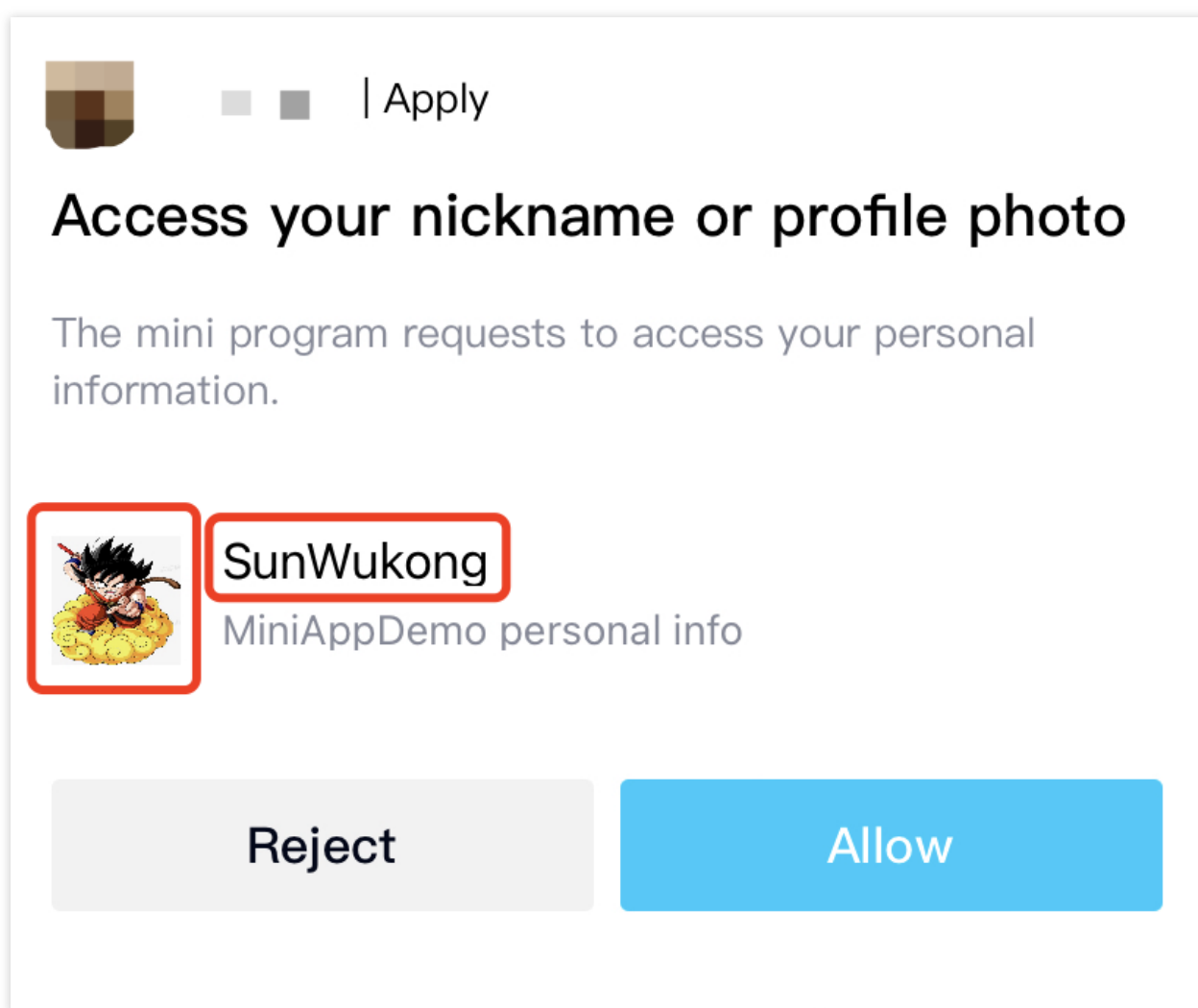
```
// 打开文档预览 - Open document preview
// @param filePath 文档路径 - document path
// @param titleName 标题 - title
// @param appInfo 小程序appinfo - Mini program appinfo
- (void)openDocument:(NSString *)filePath
                  title:(NSString *)titleName
            appInfo:(TMFMiniAppInfo *_Nonnull)appInfo;
```


定制小程序信息接口

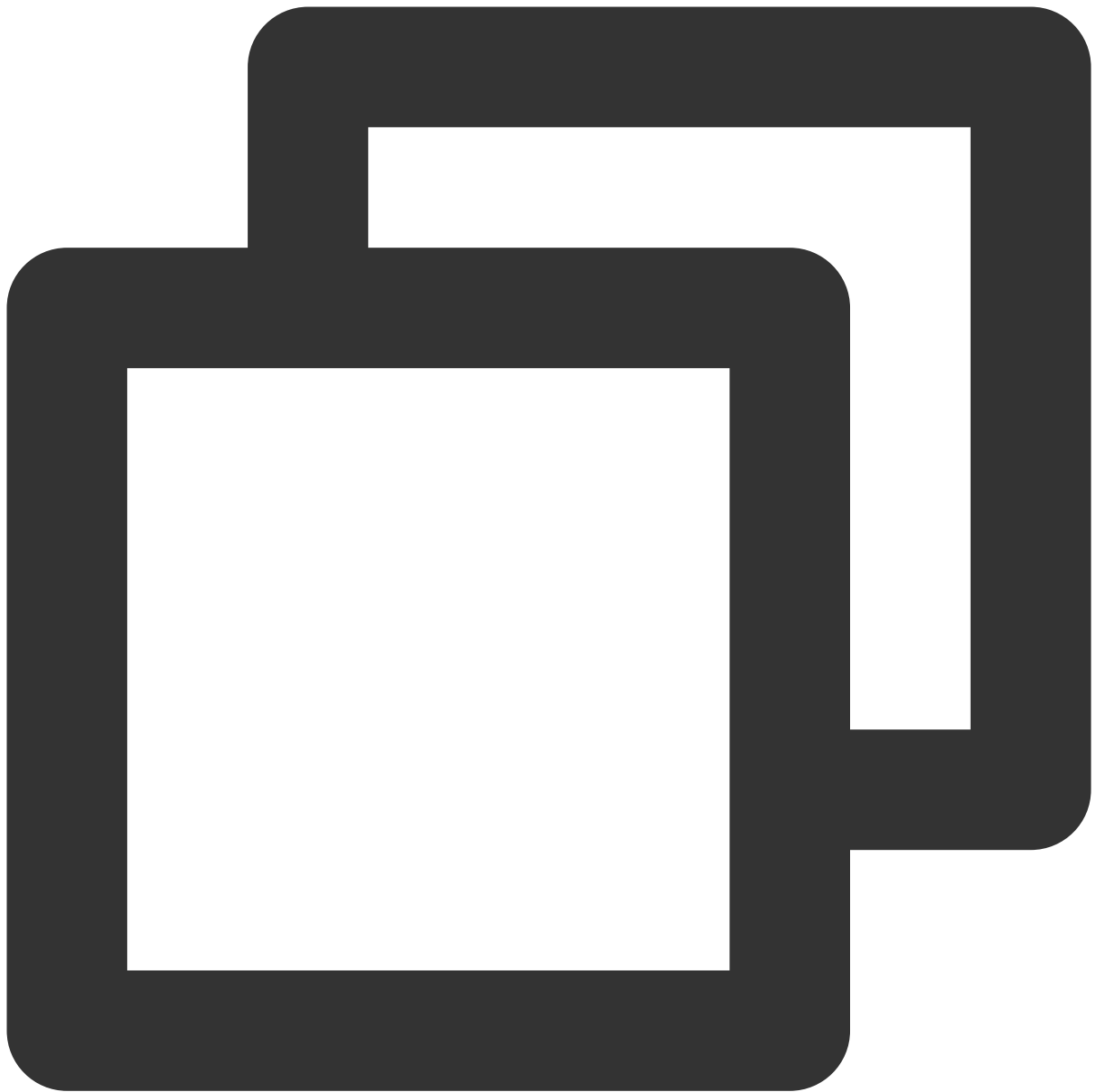
最近更新时间：2024-07-03 18:14:44

定制授权弹框中的用户信息

小程序申请获取用户信息的时候，会弹出用户授权弹框；宿主应用可以定制用户授权弹框部分信息，包括有：头像图片和用户昵称，如下图：

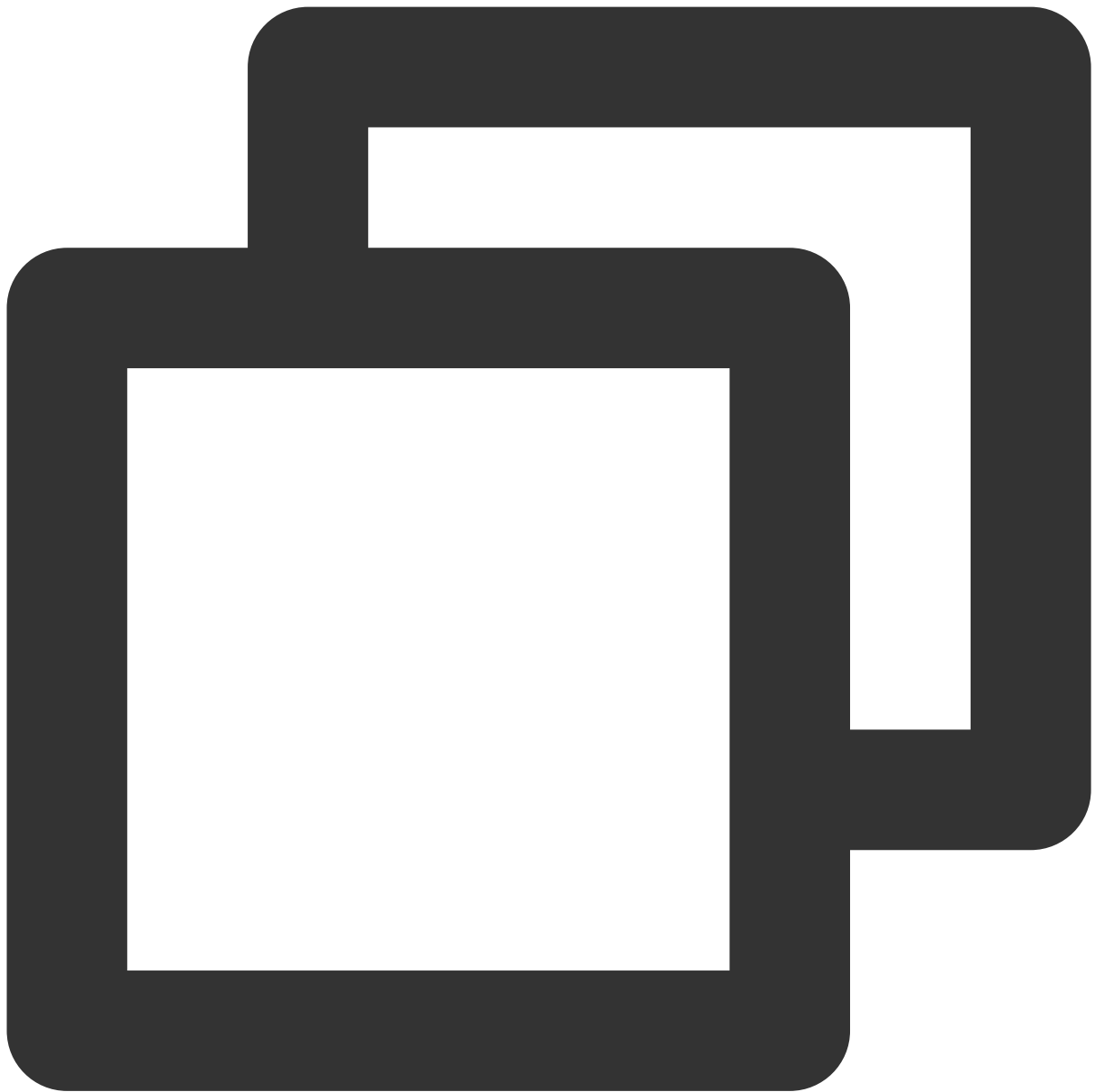


可以同过重写 `fetchAppUserInfoWithScope` 方法实现定制。



```
/**
 * @brief 根据scope拉取SDK宿主平台的用户头像和昵称，头像返回UIImageView，block必须运行在主线程!!!
 * - Pull the user avatar and nickname of the SDK host platform according to the scope
 */
- (void)fetchAppUserInfoWithScope:(NSString *)scope block:(TMAAppFetchUserInfoBlock)
```

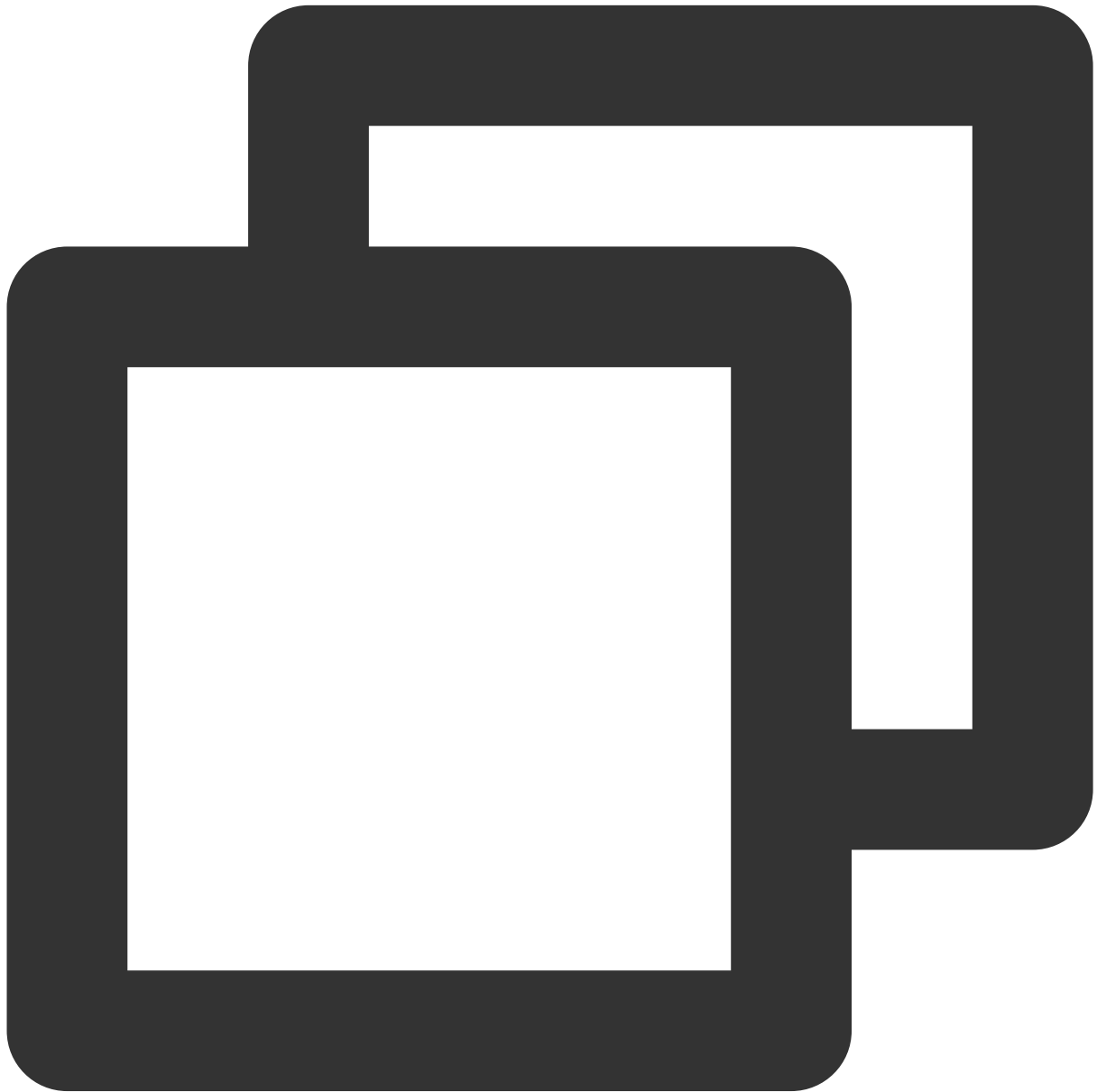
示例代码：



```
- (void)fetchAppUserInfoWithScope:(NSString *)scope block:(TMAAppFetchUserInfoBlock)block {
    if (block) {
        UIImage *defaultAvatar = [UIImage imageNamed:@"defaultAvatar"];
        UIImageView *avatarView = [[UIImageView alloc] initWithImage:defaultAvatar];
        TMAAppUserInfo *userInfo = [TMAAppUserInfo new];
        userInfo.avatarView = avatarView;
        userInfo.nickName = @"SunWukong";
        block(userInfo);
    }
}
```

定制小程序内 WebView 的 userAgent

重写 customUserAgent 方法，示例代码：



```
// 定制的UA - Customized UA
// @param defaultUserAgent 默认使用的 ua - ua used by default
- (NSString *)customUserAgent:(NSString *)defaultUserAgent;
```

定制扫码能力

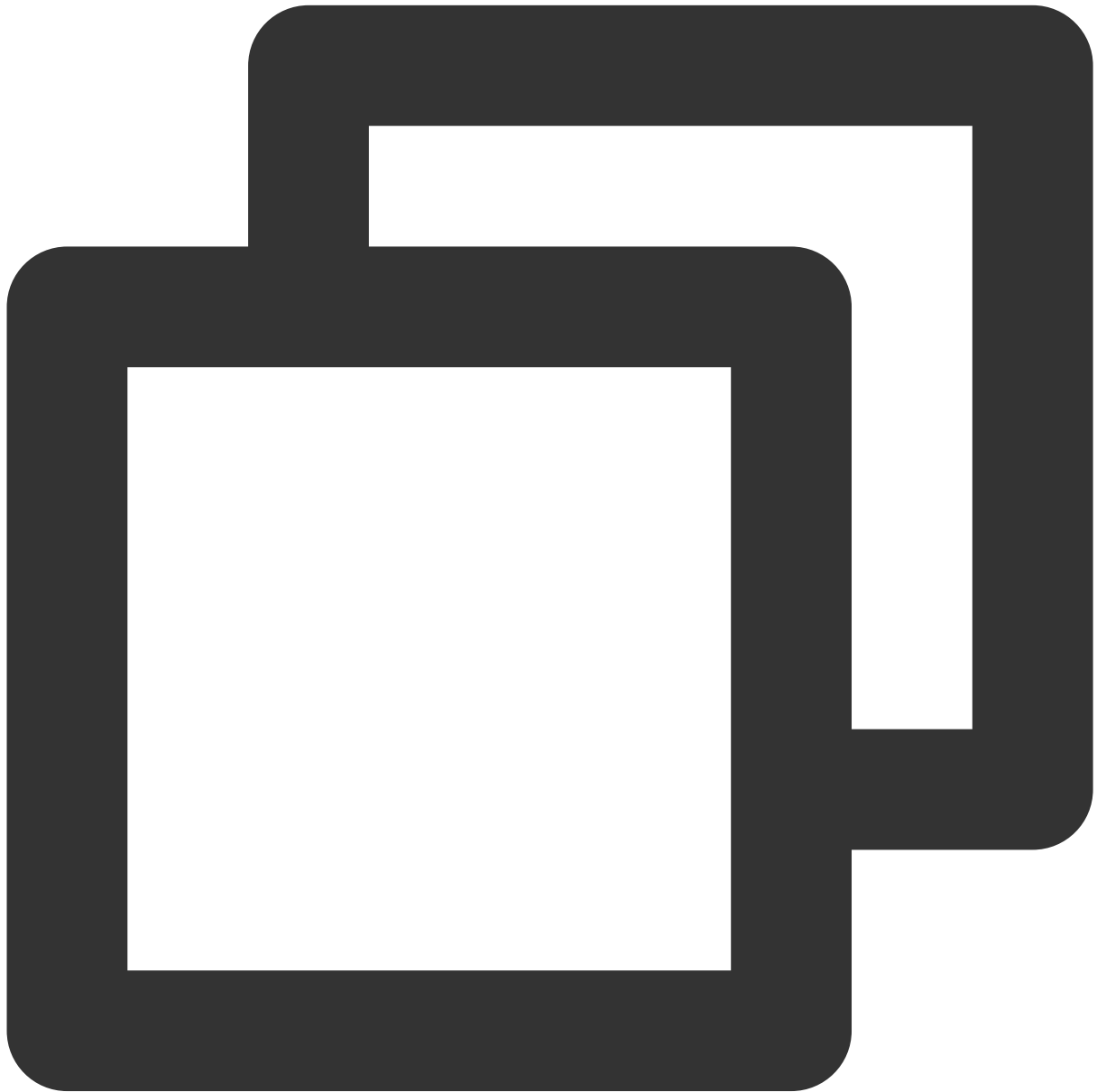
小程序 SDK 提供了默认的扫描实现（参考扩展库支持-扫一扫），也对外提供接口，供用户定制扫码能力。



```
// 扫码调用客户端的扫码模块scanCode - Scan the code to call the client's scan code module
// @param scanPrms 扫码参数字典 - scan code parameter dictionary
// @param navigationController 从那个页面呼起vc - calls vc from that page
// @param completionHandler 回调结果 - callback result
- (void)scanCode:(NSDictionary *)scanPrms
navigationController:(UINavigationController *)navigationController
completionHandler:(MACCommonCallback)completionHandler;
```

监听小程序生命周期

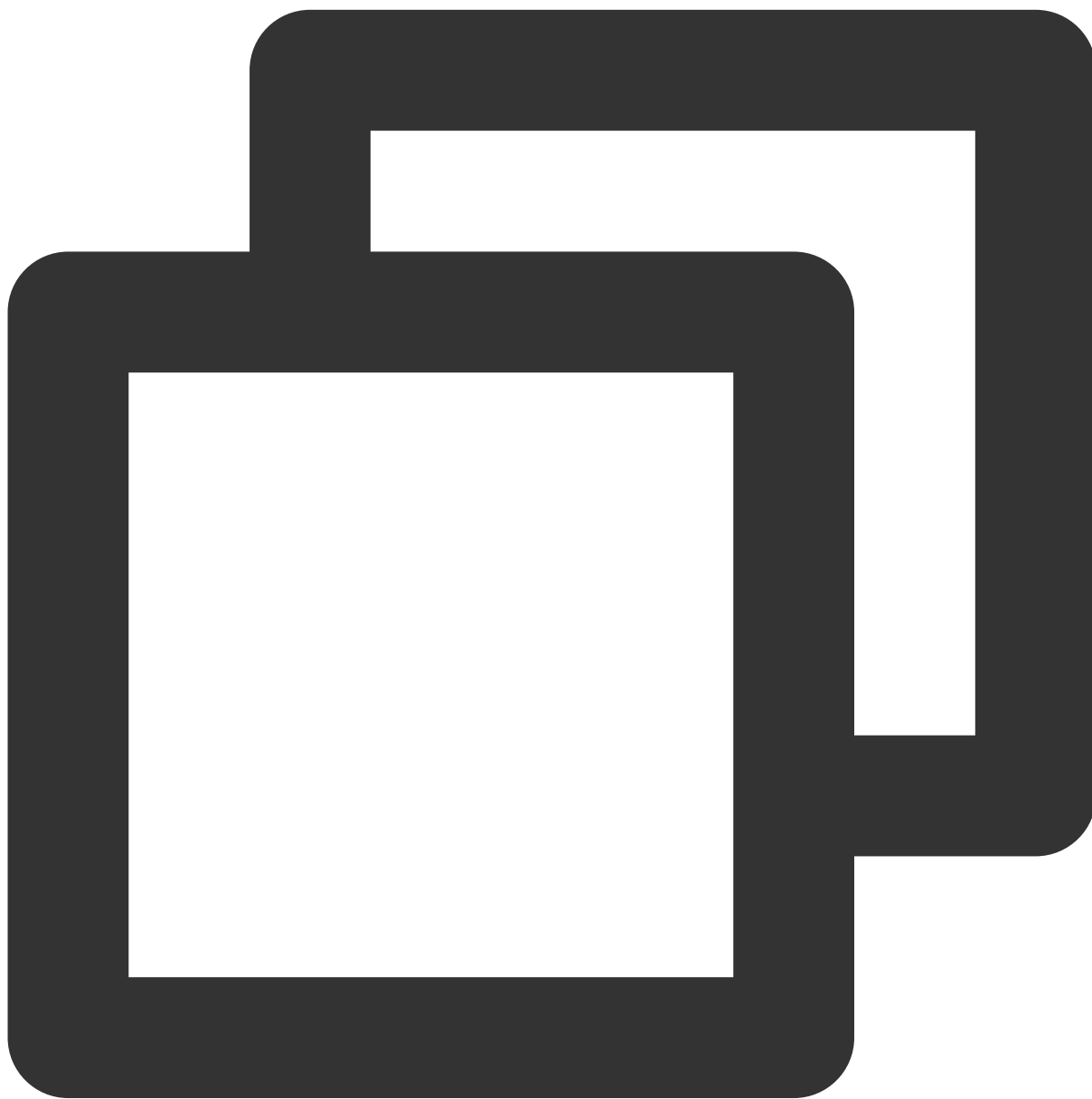
通过重写 `lifeCycleForApp` 方法，实现监听小程序生命周期。



```
// 将Application的生命周期处理透传出来，业务再去处理各自逻辑 - Transparently transmit the a
// @param appInfo 小程序信息 - The mini program information
// @param type 状态 - The state
- (void)lifeCycleForApp:(TMFMiniAppInfo *)appInfo type:(TMAAppLifeCycleType)type;
```

定制基础库更新策略和监听

通过重写 `canUpdateJSBaseLib` 方法，实现监听小程序基础库的更新信息。



```
// 基础库更新控制 - Basic library update control
// @param libInfo 更新信息 包括version:版本号, url:下载地址, md5:下载文件 md5 - The upda
// @return 是否同意更新 - Whether agree to update
- (BOOL)canUpdateJSBaseLib:(NSDictionary *)libInfo;
```

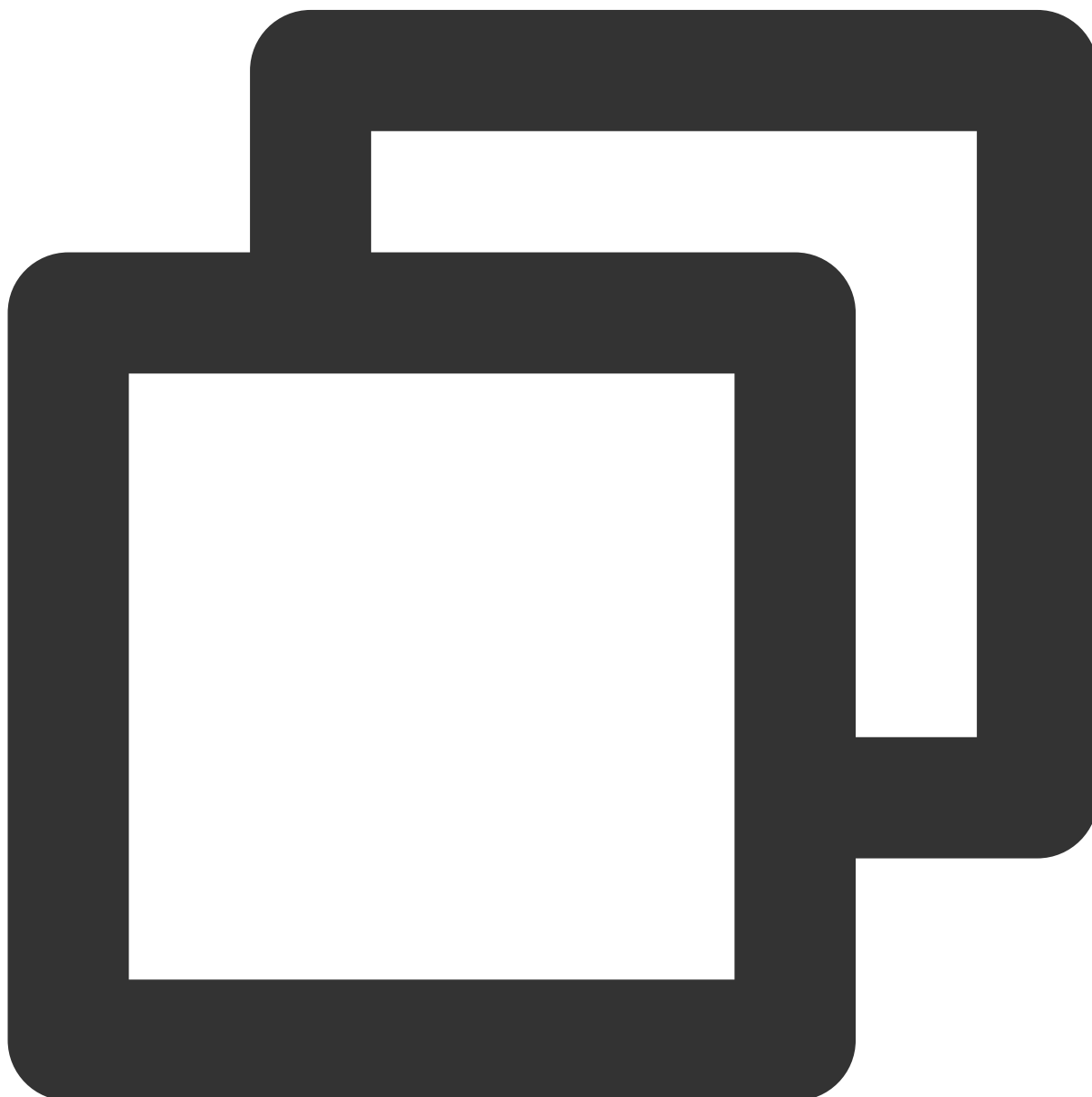

定制分享能力

最近更新时间：2024-07-24 10:03:23

默认分享渠道配置

通过重写 `defaultSharingChannels` 方法，可以自定义默认的分​​享渠道。

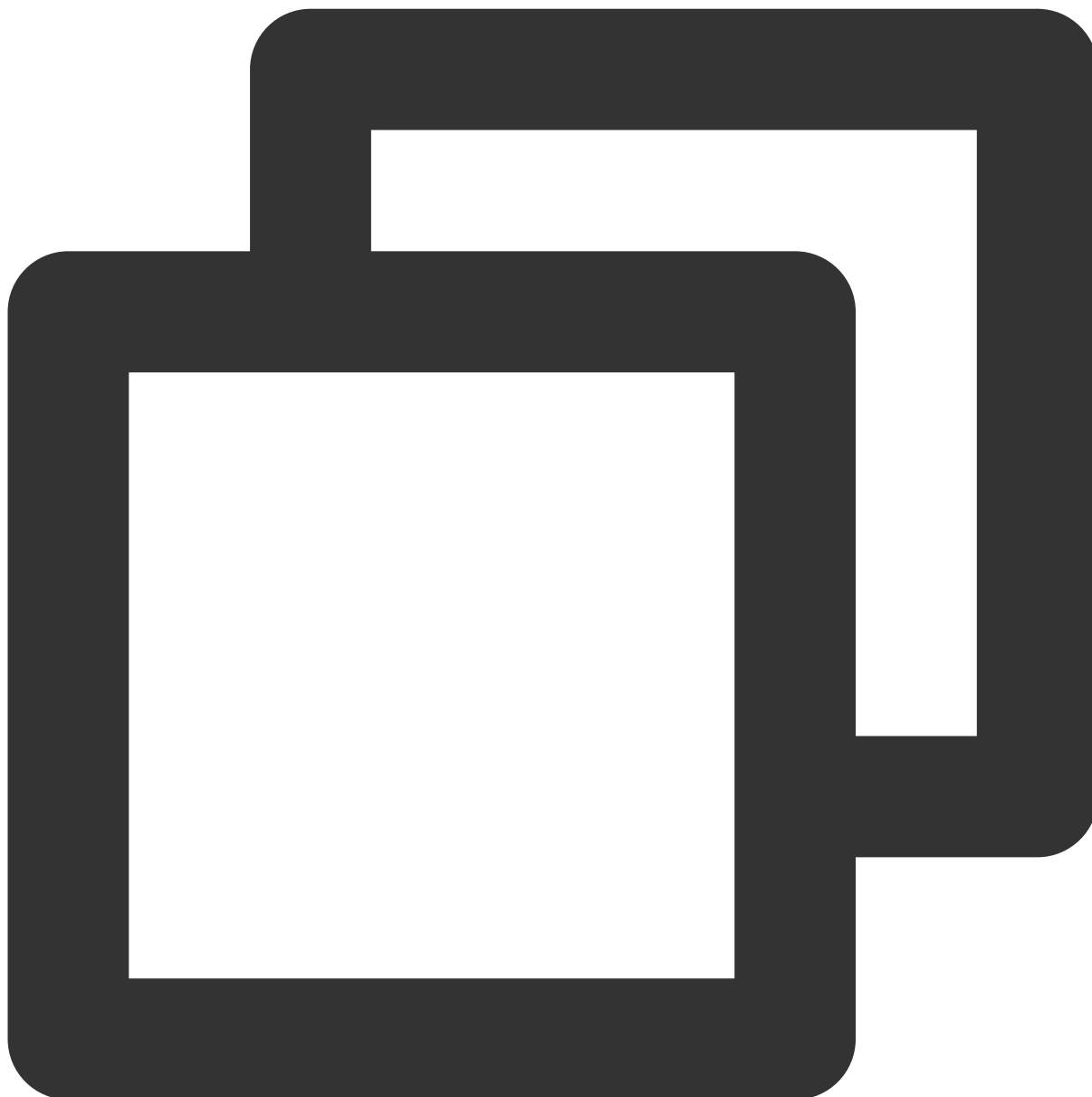
API 说明：



```
// 点击胶囊按钮后的默认分享渠道-MAShareTarget中的MAShareTargetQQ、MAShareTargetQzone、MAS  
// The default sharing channel after clicking the pill button - MAShareTarget (MASI
```

```
// (The configuration of the mini program must be a subset of the App)
- (NSArray<NSNumber *> *)defaultSharingChannels;
```

示例代码：



```
- (NSArray<NSNumber *> *)defaultSharingChannels{
    return @[@(MShareTargetQQ),@(MShareTargetWXFriends),@(MShareTargetWXMoment)]
}
```

添加自定义分享按钮

通过重写 `customizedConfigForShare` 方法，可以自定义分享途径、决定展示顺序。

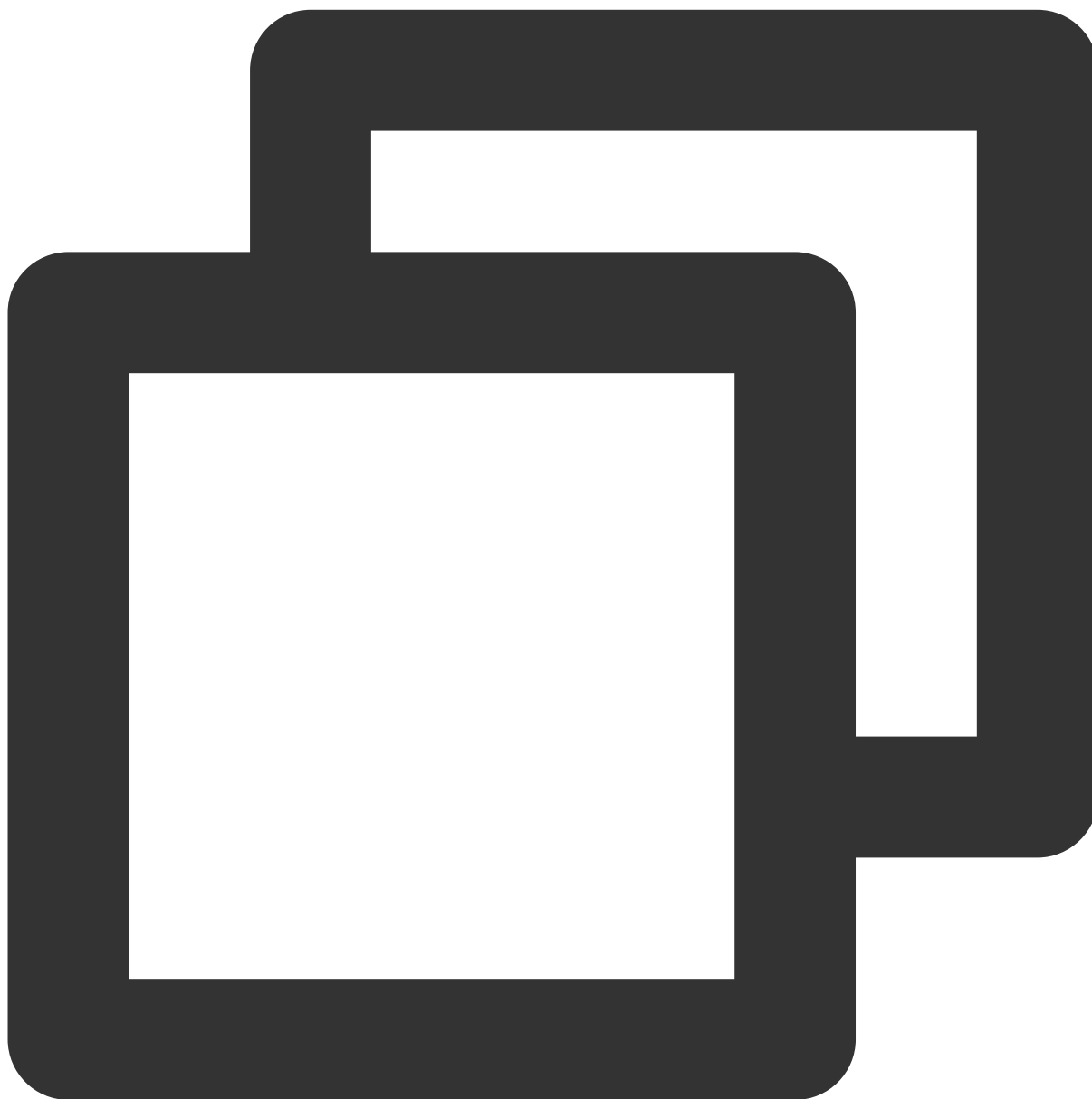
API 说明：



```
// 宿主App可以自定义分享途径、决定展示顺序，目前使用在点击更多按钮、button组件(open-type="share")
// 1、默认渠道：QQ好友、QQ空间、微信、朋友圈（具体type参见MAUIDelegateShareViewType），由开发
// 2、自定义分享渠道：宿主App自定义（type填MAUIDelegateShareViewTypeCustomizedShare，自定义
// 3、自定义事件处理：宿主App自定义（type填MAUIDelegateShareViewTypeCustomizedAction）
// 以上三种渠道展示顺序支持混排
///
// The host App can customize the sharing path and determine the display order. It
```

```
// 1. Default channels: QQ Friends, QQ Space, WeChat, Moments (for specific types,  
// 2. Customized sharing channel: Host App customization (type fills in MAUIDelegat  
// 3. Custom event processing: Host App customization (type fills in MAUIDelegateSh  
// The display order of the above three channels  
- (NSArray<TMASheetItemInfo *> *)customizedConfigForShare;
```

示例代码：

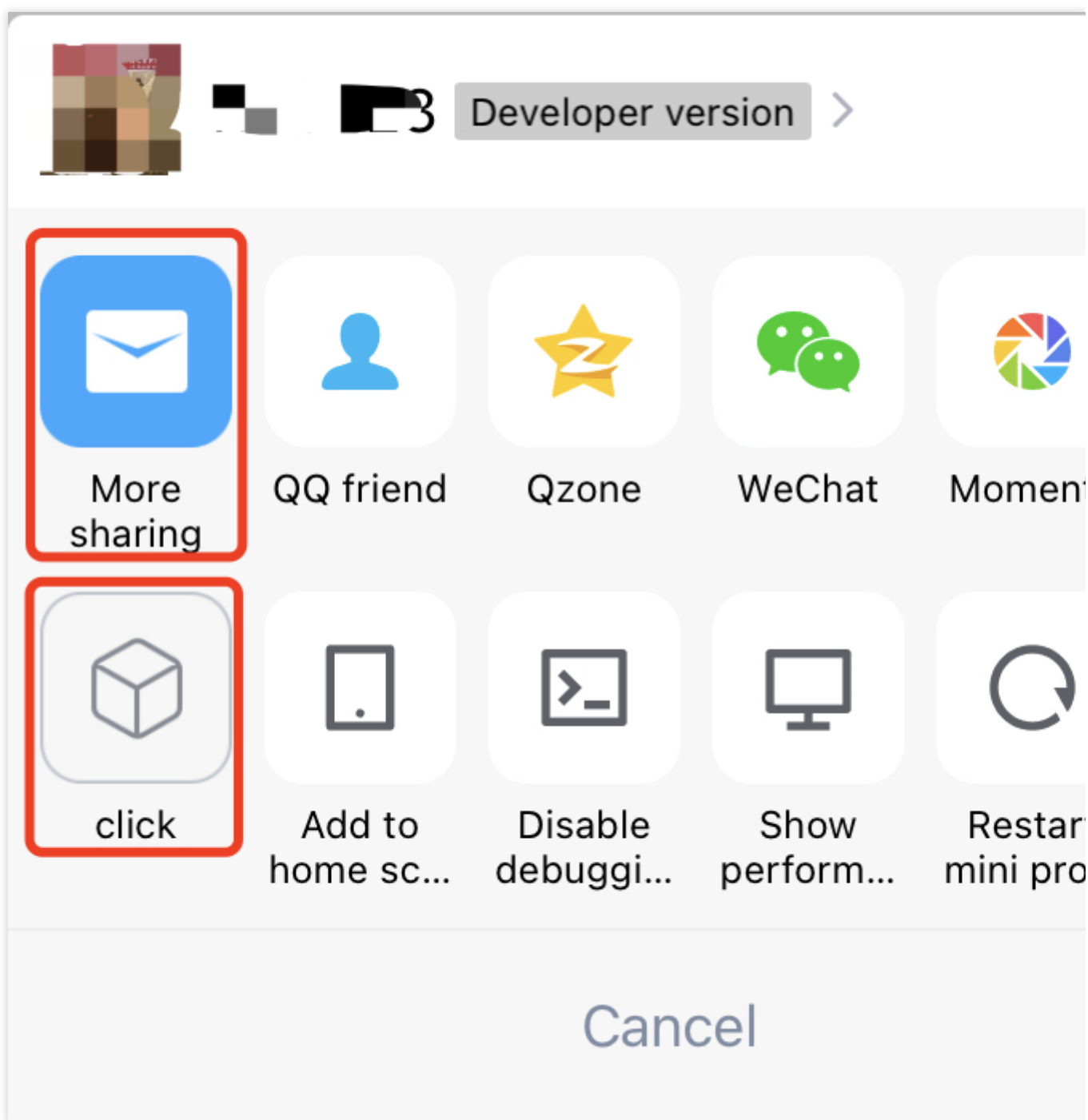


```
- (NSArray<TMASheetItemInfo *> *)customizedConfigForShare {  
    NSMutableArray *arrays = [[NSMutableArray alloc] init];  
    TMASheetItemInfo *item1 = [[TMASheetItemInfo alloc] initWithTitle:@"More sharin
```

```
item1.icon = [UIImage imageNamed:@"icon_moreOperation_shareChat"];
item1.shareTarget = 10001;
[arrays addObject:item1];

TMASheetItemInfo *item2 = [[TMASheetItemInfo alloc] initWithTitle:@"click" type
    NSLog(@"click 点击");
]];
item2.icon = [UIImage imageNamed:@"icon_moreOperation_collect"];
[arrays addObject:item2];
return arrays;
}
```

效果如下：

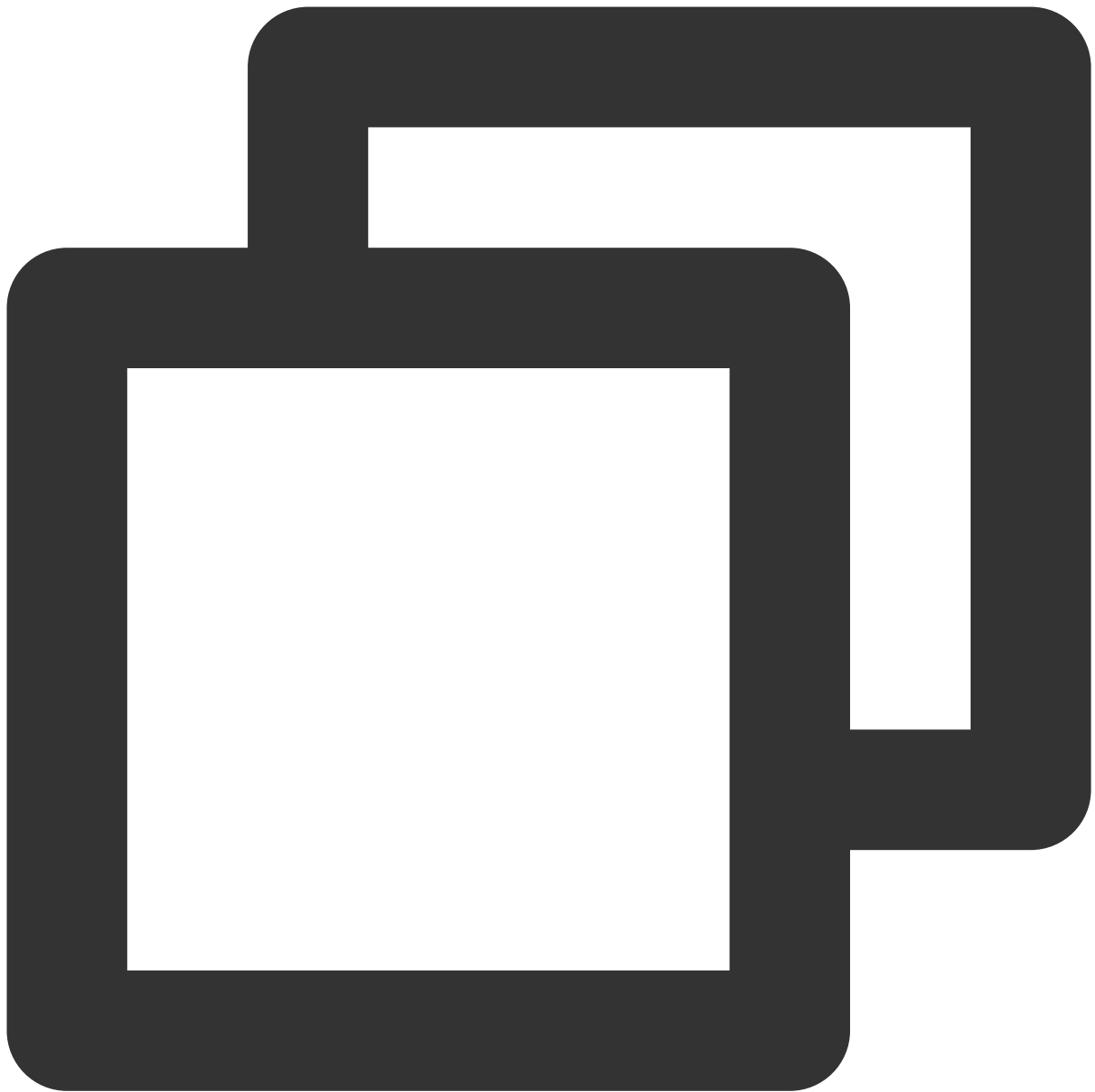


小程序内部逻辑

小程序内部通过 `onShareAppMessage` 监听用户单击分享类按钮的行为，并自定义转发内容。

分享逻辑实现

宿主 App 收到小程序内部返回的分享数据后，触发 `shareMessageWithModel` 执行最终的分享动作，开发者可以根据 `shareTarget` 等数据，自行处理需要分享的内容，对接三方分享平台。



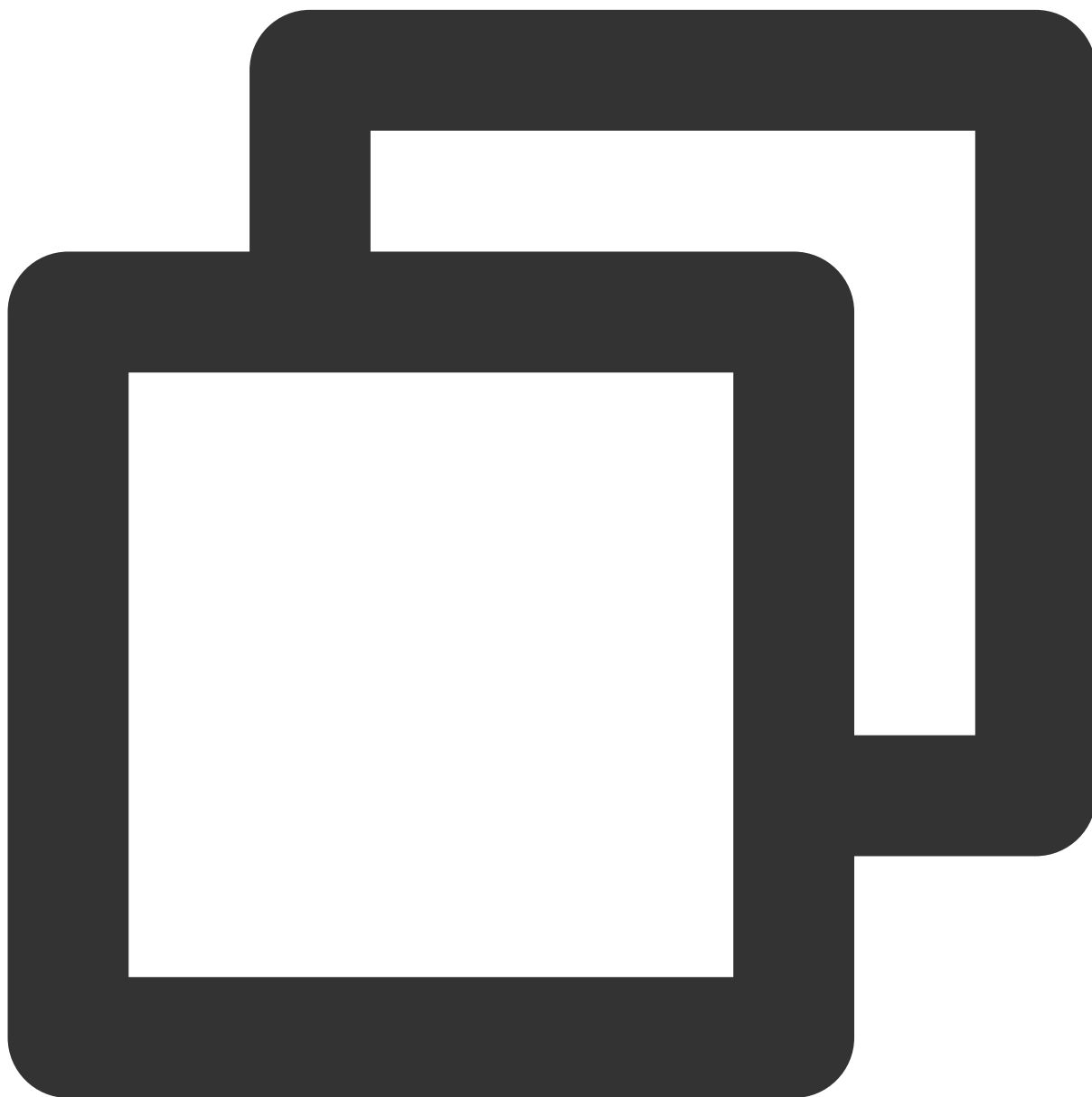
```
/**
 * @brief 分享的接口 - shared interface
 * @param shareModel 分享model - share model
 * @param appInfo 小程序appinfo - Mini program appinfo
 * @param completionBlock 回调 - callback
 */
- (void)shareMessageWithModel:(TMAShareModel *_Nonnull)shareModel
                        appInfo:(TMFMiniAppInfo *_Nonnull)appInfo
                        completionBlock:(nullable void (^)(NSError * _Nullable error))completi
```

定制用户属性

最近更新时间：2024-07-03 18:14:36

宿主应用开发者可以通过 SDK 提供的方法设置用户的昵称和头像信息。

设置用户昵称

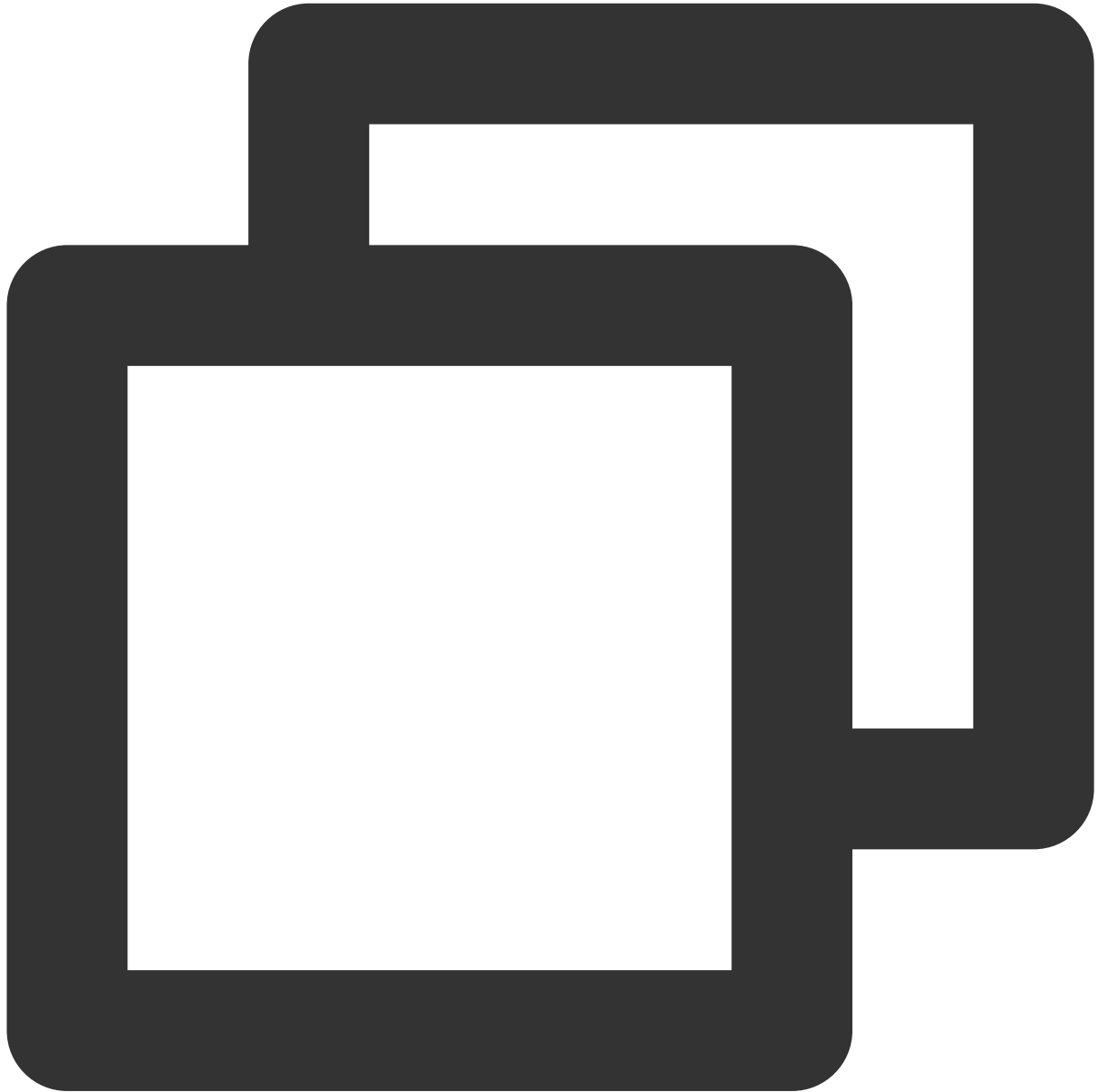


```
/**
```

```
 * @brief SDK宿主平台的用户昵称,默认返回"TCMPP" - The user nickname of the SDK host plat
```

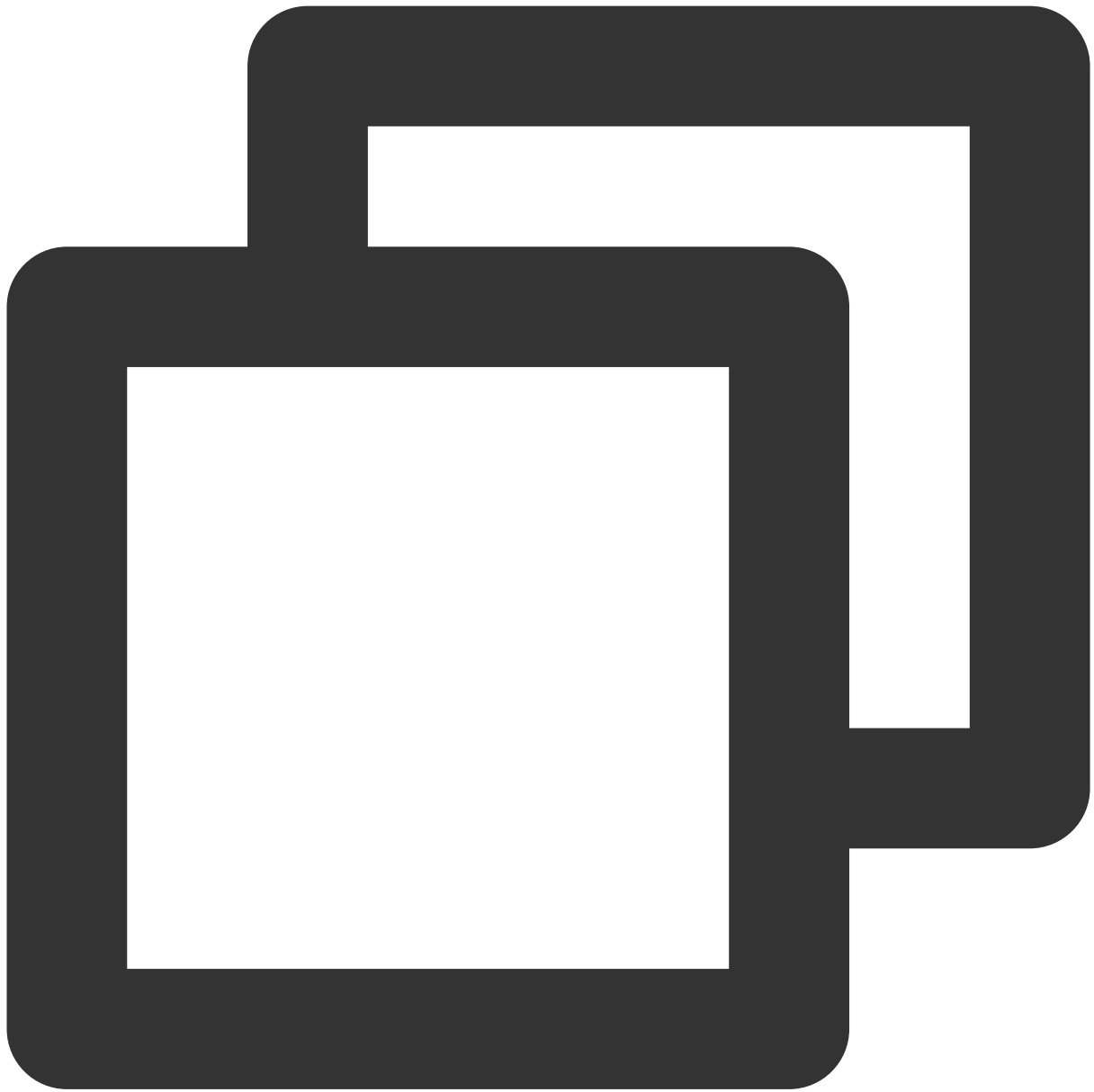
```
*/  
- (NSString *)getAppNickName;
```

设置用户头像



```
/**  
 * @brief SDK宿主平台的用户头像,默认返回demo图片 - SDK host platform user avatar, return  
 */  
- (UIImage *)getAppAvatar;
```

获取当前用户账号标识



```
/**
 * @brief 获取SDK宿主平台的当前用户账号标识，一般填uin或openid - Get the current user account
 *
 * 注意：返回nil会导致SDK内某些缓存失效。如果没有登录，可以填个设备号id来避免缓存失效 - Note: Return nil will cause some cache in SDK to be invalid. If not logged in, you can fill in a device ID to avoid cache invalidation.
 */
- (NSString *_Nonnull) getAppUID;
```


日志与事件上报

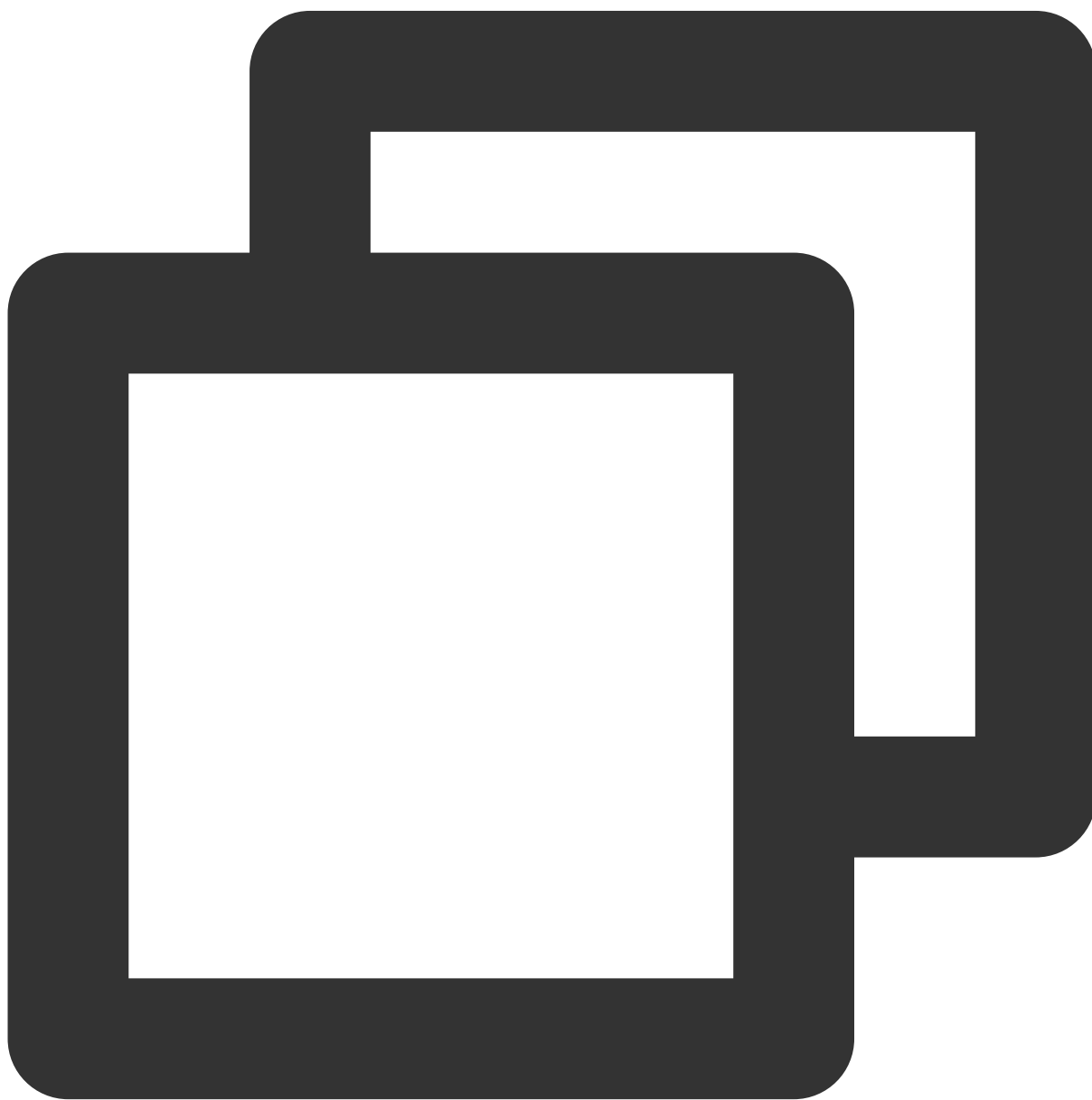
最近更新时间：2024-07-03 18:07:10

SDK 运行日志打印

在开发阶段实现日志输出接口，方便进行问题排查。

Note：

APP 发布上线时，建议关闭日志输出，保证安全性及性能。

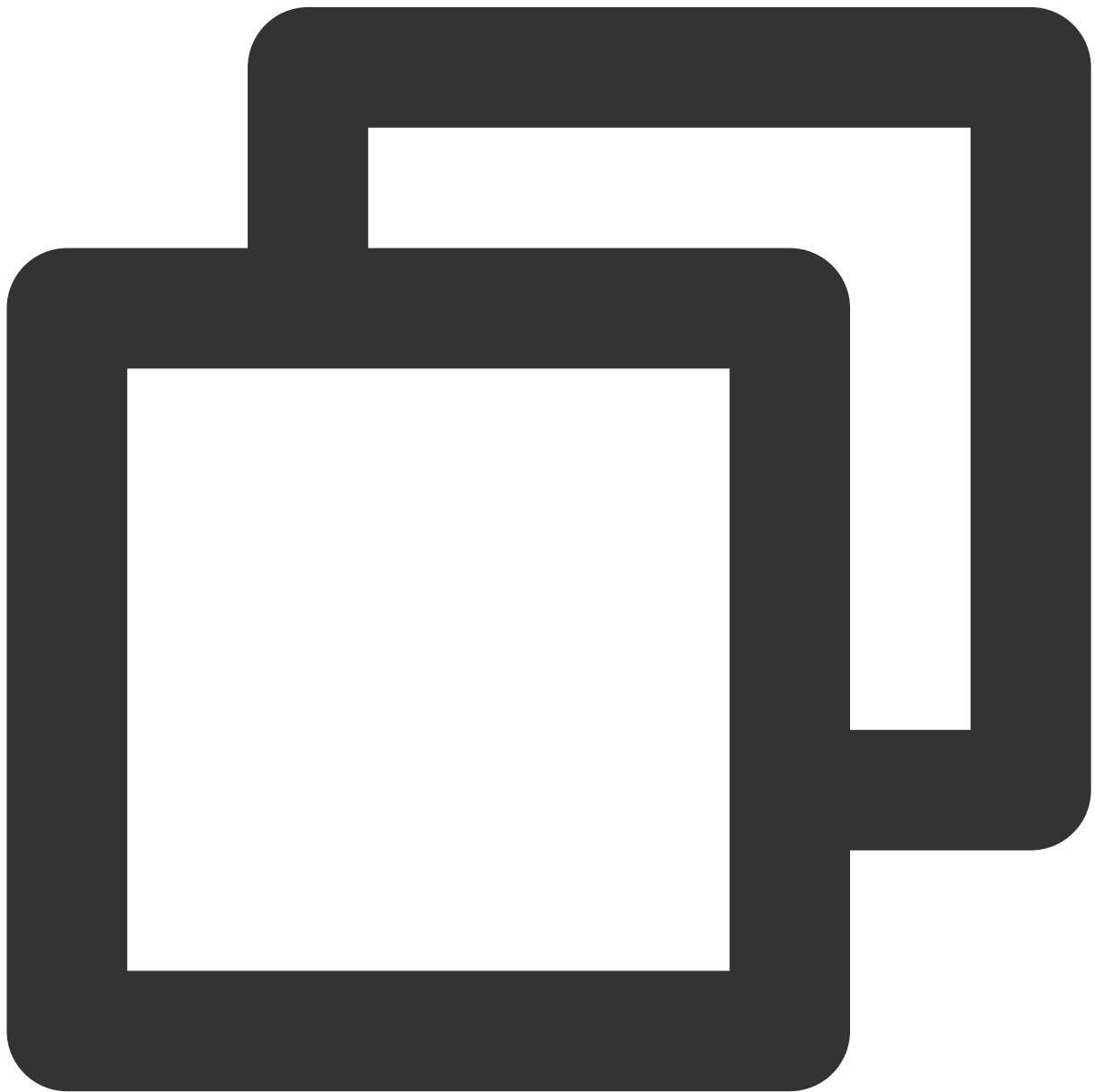


```
/// 打印Log  
/// @param level log级别, 参考PLTLogLevel  
/// @param msg log信息  
- (void)log:(MALogLevel)level msg:(NSString *)msg;
```

事件上报

宿主 APP 可以实现事件上报接口, 来覆盖 TCMPP SDK 内部的上报逻辑。

包括小程序操作事件及小程序内部调用 `wx.reportEvent` 上报的数据。



```
typedef NS_ENUM(NSInteger, TMAReportEventID) {
    // 未定义, 报告用户自定义事件
    // Undefined, report user-defined events
    TMAReportEventID_None = 0,
    // 打开小程序
    // Open the mini app
    TMAReportEventID_OPEN_MINIAPP = 1,
    // 更新小程序
    // Update the mini app
    TMAReportEventID_UPDATE_MINIAPP = 2,
    // 下载小程序
    // Download the mini app
    TMAReportEventID_DOWNLOAD_MINIAPP = 3,
    // 小程序页面浏览
    // Open the mini app page
    TMAReportEventID_MINIAPP_PAGE_VIEW = 4,
    // 退出小程序
    // Close the mini app
    TMAReportEventID_EXIT_MINIAPP = 5,
    // 小程序行为事件, atcion:0 后台;1 前台
    // Mini program action events, atcion: 0 onHide; 1 onShow
    TMAReportEventID_MINIAPP_ACTION = 6
};

// 上报数据 - Report data
// @param event 事件, 参考TMAReportEventID - Event Id, Reference TMAReportEventID
// @param eventName 事件名称 - Event Name
// @param params 参数 - params
// @param appInfo 小程序信息 - Mini program info
// @return 是否拦截内部上报 - Whether to intercept internal reports
- (BOOL)reportEvent:(int)eventId
    eventName:(NSString *)eventName
    params:(NSDictionary *)params
    appinfo:(TMFMiniAppInfo *)appInfo;
```

小程序实时日志上报

宿主 APP 可以实现事件实时日志上报接口，来覆盖 TCMPP SDK 内部的上报逻辑。

包括小程序内部调用 `wx.getRealtimeLogManager` 写入的日志数据。



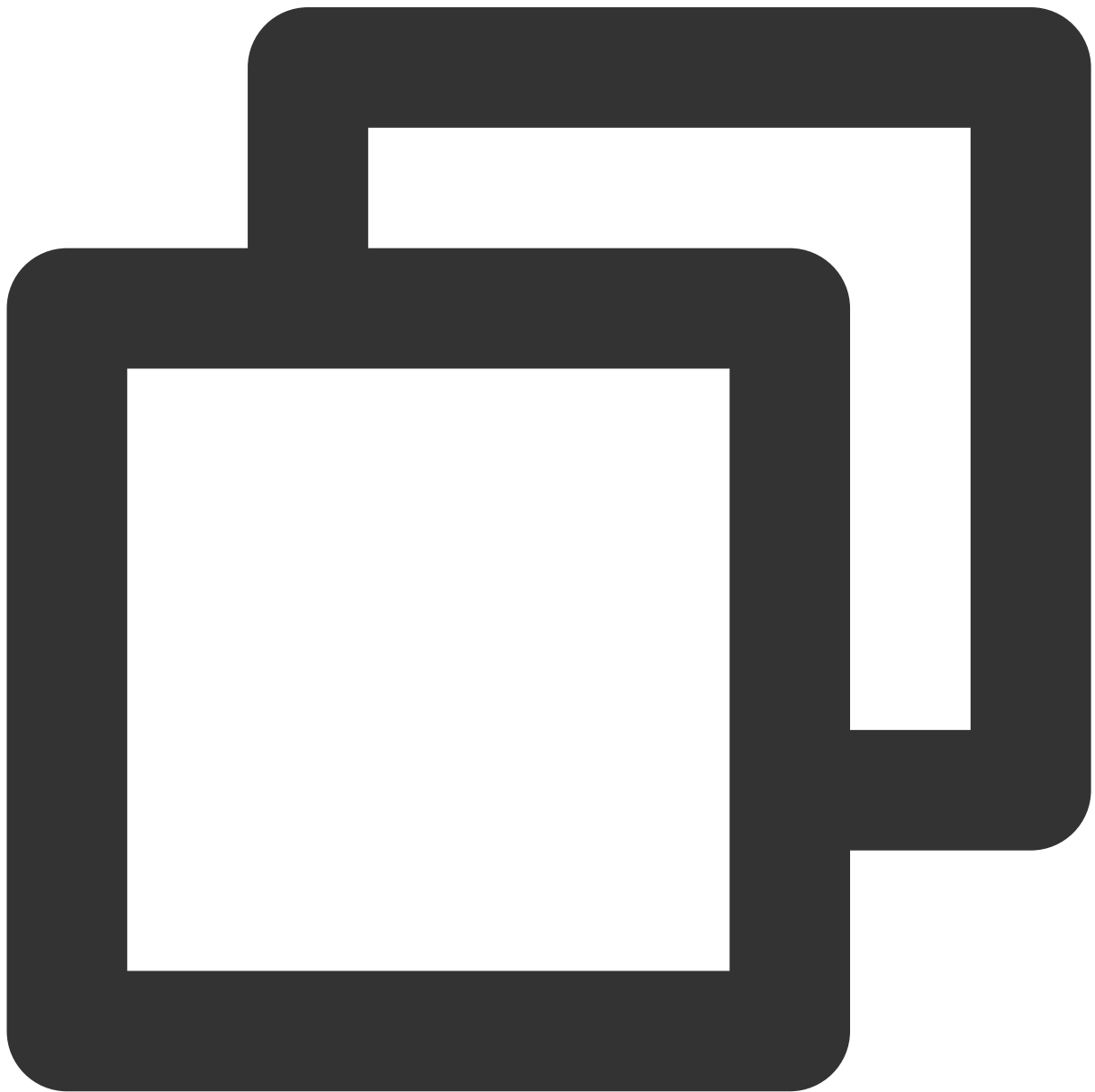
```
// 上报日志数据 - Report log data
// @param appId appID
// @param jsVersion 基础库版本 - The basic library version
// @param page 当前页面 - The current page
// @param filterMsgs 过滤内容 - The filter contents
// @param logs 日志事件 - Log events
// @param completionBlock 结果回调 - Callback result
// @return 是否拦截内部上报 - Whether to intercept internal reports
- (BOOL)reportRealTimeLogWithAppId:(NSString *)appId
                        jsVersion:(NSString *)jsVersion
                        page:(NSString *)page
```

```
filterMsgs:(NSArray <NSString *>*)filterMsgs
logs:(NSArray <TMARealtimeLogItem *>*)logs
completionBlock:(void (^)(NSError * _Nullable error))completionB
```

小程序内部日志上报

宿主 APP 可以实现事件实时日志上报接口，来覆盖 TCMPP SDK 内部的上报逻辑。

包括小程序内部调用 `wx.getLogManager` 写入的日志数据，用户可以通过使用 `button` 组件的 `open-type="feedback"` 来上传打印的日志。



```
// 上传appID对应小程序的log - Upload the log of the applet corresponding to the appID
// 实现时，可以使用TMFMiniAppSDKManager的`sandBoxPathWithAppID:`接口获取沙箱路径，然后拼接
// @param appID 小程序的appID - The appID of mini program/mini game

- (void)uploadLogFileWithAppID:(NSString *)appID;
```

开放接口

最近更新时间：2024-07-03 18:06:54

微信小程序开放接口如下方法与 可以通过代理类实现，由 App 实现具体的逻辑并返回给小程序。

小程序方法：

wx.login

wx.getUserInfo

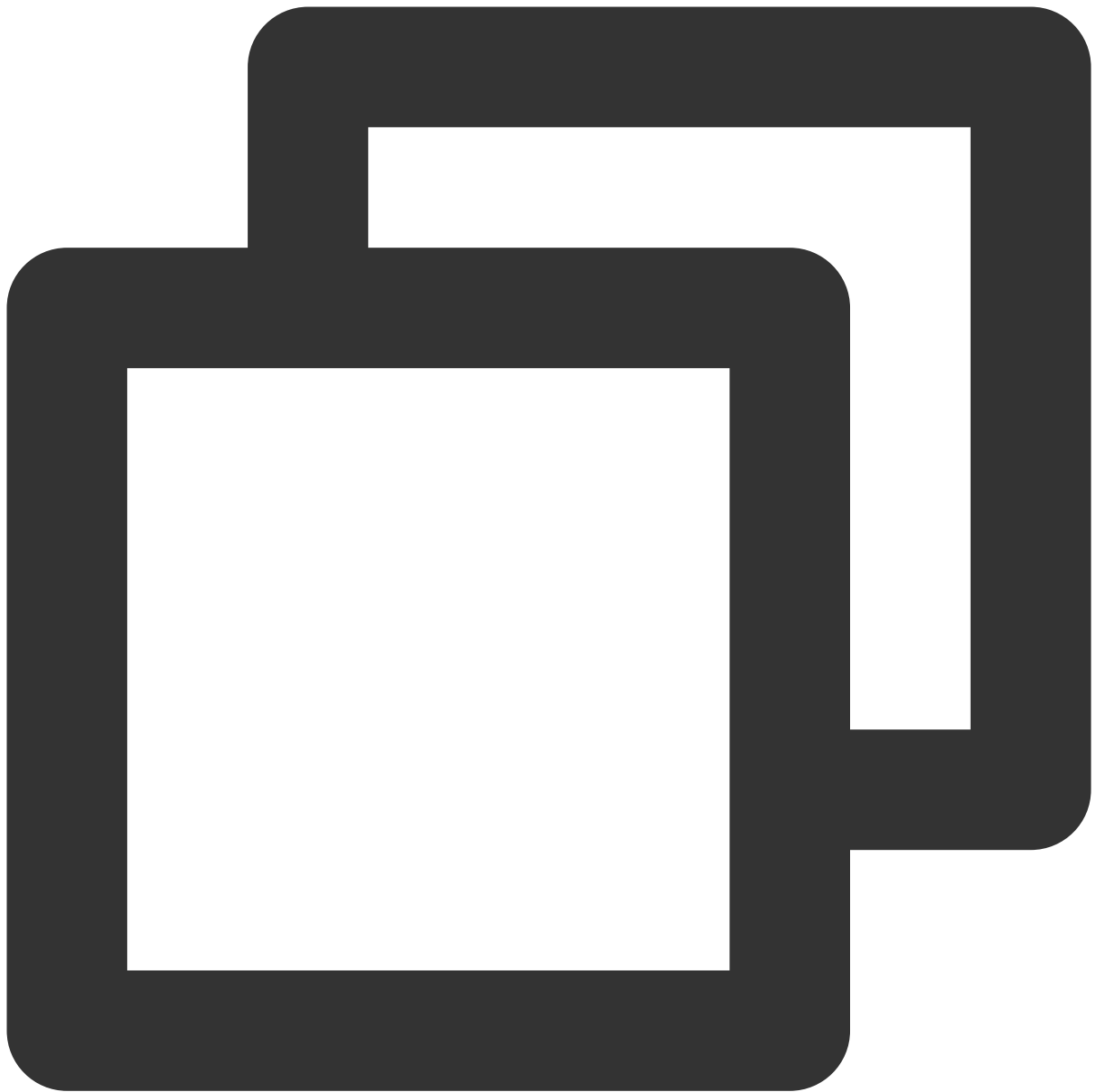
wx.getUserProfile

wx.checkSession

wx.requestPayment

wx.getPhoneNumber

代理类中相关接口



```
// 发起支付 - Request payment

// @param app 小程序/小游戏实例 - Mini program/mini game instance

// @param params 参数 - parameters

// @param completionHandler 回调结果 - Result callback

- (void)requestPayment:(TMFMiniAppInfo *)app params:(NSDictionary *)params completi
```



```
// login

// @param app 小程序/小游戏实例 - Mini program/mini game instance

// @param params 参数 - parameters

// @param completionHandler 回调结果 - Result callback

- (void)login:(TMFMiniAppInfo *)app params:(NSDictionary *)params completionHandler

// checkSession

// @param app 小程序/小游戏实例 - Mini program/mini game instance

// @param params 参数 - parameters

// @param completionHandler 回调结果 - Result callback

- (void)checkSession:(TMFMiniAppInfo *)app params:(NSDictionary *)params completion

// getUserProfile

// @param app 小程序/小游戏实例 - Mini program/mini game instance

// @param params 参数 - parameters

// @param completionHandler 回调结果 - Result callback

- (void)getUserProfile:(TMFMiniAppInfo *)app params:(NSDictionary *)params completi

// getPhoneNumber

// @param app 小程序/小游戏实例 - Mini program/mini game instance

// @param params 参数 - parameters

// @param completionHandler 回调结果 - Result callback

- (void)getPhoneNumber:(TMFMiniAppInfo *)app params:(NSDictionary *)params completi
```

```
// getUserInfo

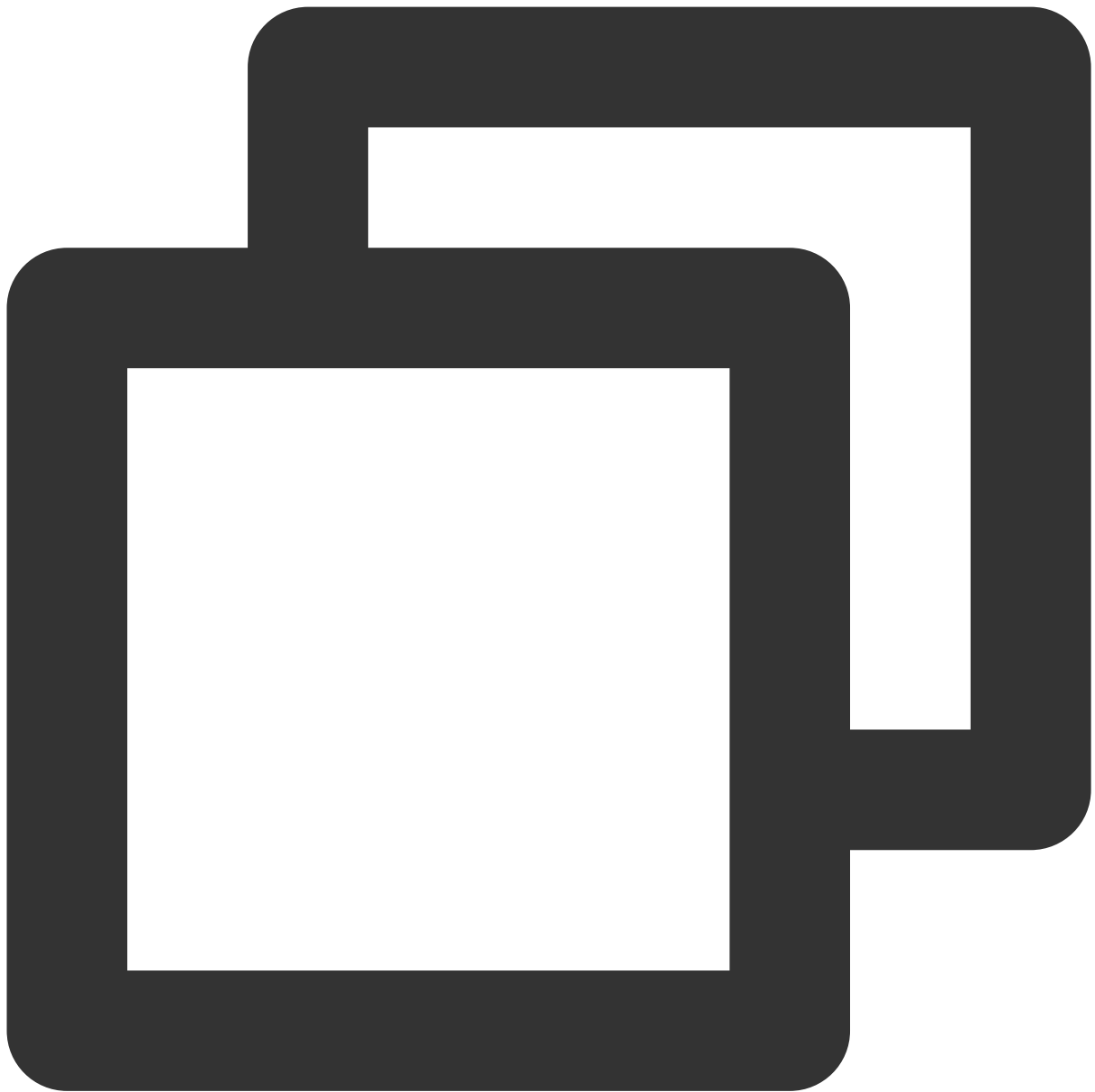
// @param app 小程序/小游戏实例 - Mini program/mini game instance

// @param params 参数 - parameters

// @param completionHandler 回调结果 - Result callback

- (void)getUserInfo:(TMFMiniAppInfo *)app params:(NSDictionary *)params completionHandler
```

实例代码



```
// 发起支付 - Request payment

// @param app 小程序/小游戏实例 - Mini program/mini game instance

// @param params 参数 - parameters

// @param completionHandler 回调结果 - Result callback

- (void)requestPayment:(TMFMiniAppInfo *)app params:(NSDictionary *)params completi
```

```
// login

// @param app 小程序/小游戏实例 - Mini program/mini game instance

// @param params 参数 - parameters

// @param completionHandler 回调结果 - Result callback

- (void)login:(TMFMiniAppInfo *)app params:(NSDictionary *)params completionHandler

// checkSession

// @param app 小程序/小游戏实例 - Mini program/mini game instance

// @param params 参数 - parameters

// @param completionHandler 回调结果 - Result callback

- (void)checkSession:(TMFMiniAppInfo *)app params:(NSDictionary *)params completion

// getUserProfile

// @param app 小程序/小游戏实例 - Mini program/mini game instance

// @param params 参数 - parameters

// @param completionHandler 回调结果 - Result callback

- (void)getUserProfile:(TMFMiniAppInfo *)app params:(NSDictionary *)params completi

// getPhoneNumber

// @param app 小程序/小游戏实例 - Mini program/mini game instance

// @param params 参数 - parameters

// @param completionHandler 回调结果 - Result callback

- (void)getPhoneNumber:(TMFMiniAppInfo *)app params:(NSDictionary *)params completi
```

```
// getUserInfo

// @param app 小程序/小游戏实例 - Mini program/mini game instance

// @param params 参数 - parameters

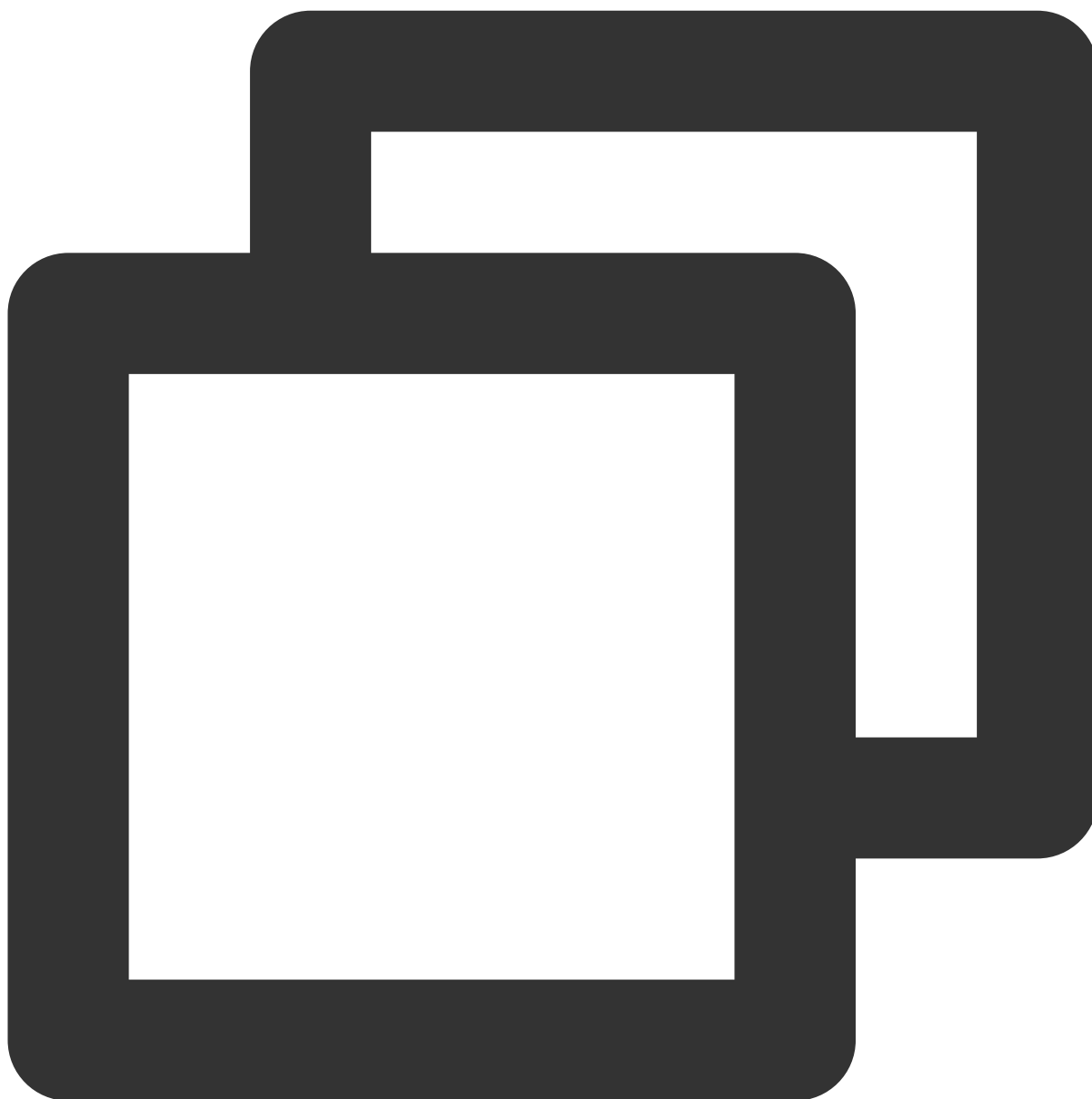
// @param completionHandler 回调结果 - Result callback

- (void)getUserInfo:(TMFMiniAppInfo *)app params:(NSDictionary *)params completionHandler
```

其他

最近更新时间：2024-08-21 09:40:05

宿主 App 相关接口



```
/**
 * @brief SDK宿主应用名称 - Host app name
 * 主要用于文案提示使用 - Mainly used for copywriting prompts
 */
```

```
- (NSString *)appName;

/**
 * @brief SDK宿主平台版本 - Host app version
 * @return 返回的是小写字符串, 例如1.0.0 - returns a lowercase string, such as 1.0.0
 */
- (NSString *)getAppVersion;

/**
 * @brief SDK宿主平台的网络状态 - The network status of the host platform
 */

- (TMANetWorkStatus)getAppNetworkStatus;

/**
 * @brief SDK宿主平台的机型信息 - SDK host platform model information
 */
- (NSString *)getAppIPhoneModel;

/**
 * @brief SDK宿主平台设备信息 - SDK host platform device information
 * @return 格式 - format : @{@"brand":@"iPhone",@"model":@"iPhone 11<iPhone12,1>",@"s
 */
- (NSDictionary *)getAppDeviceInfo;

/**
 * @brief 获取宿主APP基础信息 - Get basic information of the host APP
 * @return 格式 - format : @{@"SDKVersion":@"2.32.2",@"model":@"iPhone 11<iPhone12,1>
 */
- (NSDictionary *)getAppBaseInfo;

/**
 * @brief 宿主设置的当前语言 - Get current language set by the app
 * @return 格式 - format : "zh-Hans"
 */
- (NSString *)getCurrentLocalLanguage;

/**
 * @brief 宿主设置的当前主题 如果不实现此方法 那么getAppBaseInfo中返回的theme为系统主题 - The
 */
- (NSString *)getAppTheme;

/**
 * @brief 剪贴板频控 - Clipboard frequency control
 */
- (NSNumber *)getClipboardInterval;
```

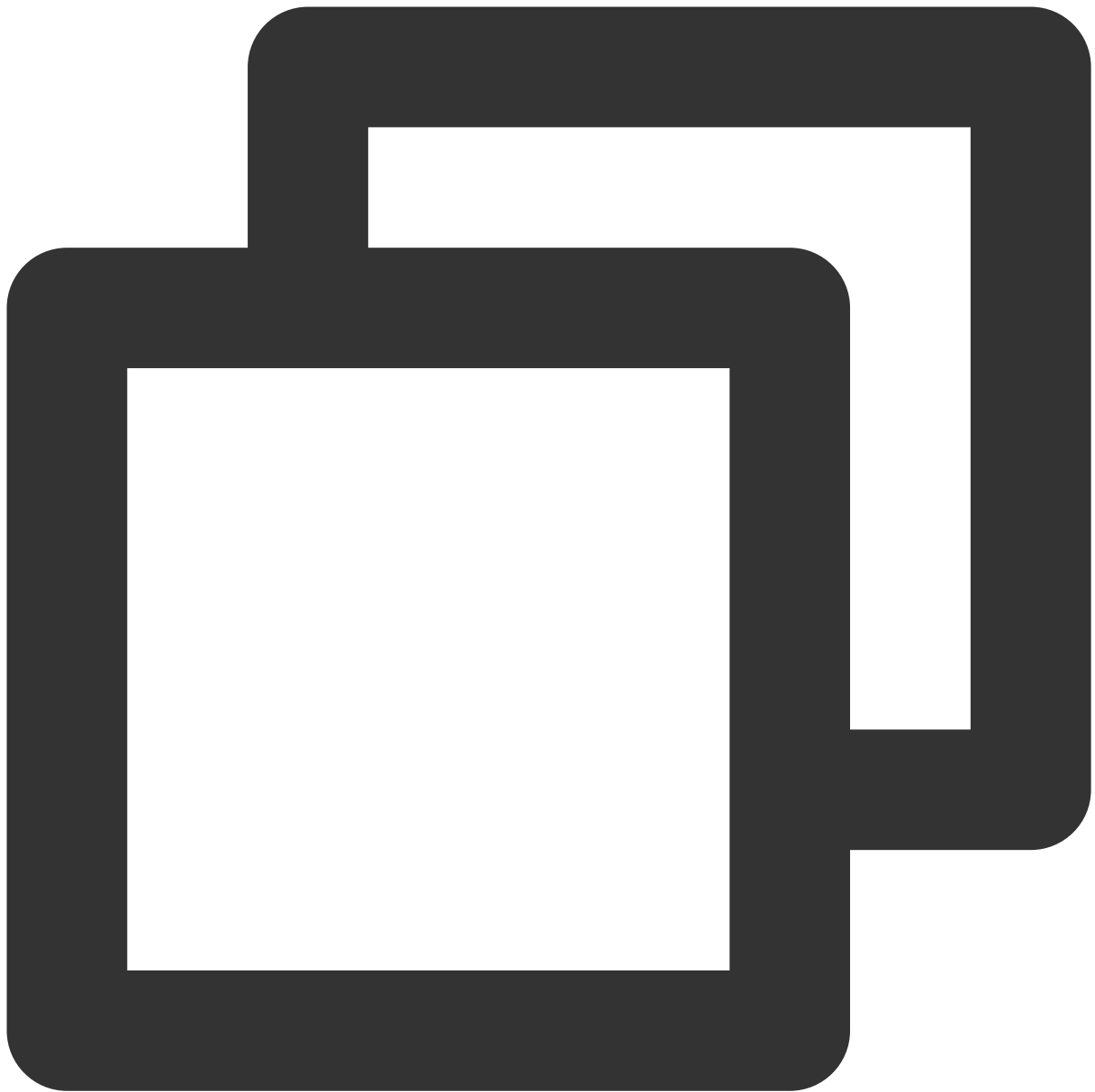
```
// 小程序最多保活的个数,默认是3

// The maximum number of mini programs that can be kept alive, the default is 3

- (NSInteger)maxMiniAppKeepAliveCount;


// 设置桌面快捷方式宿主URL Scheme
// Set desktop shortcut App URL Scheme
- (NSString *)getAppScheme;
```

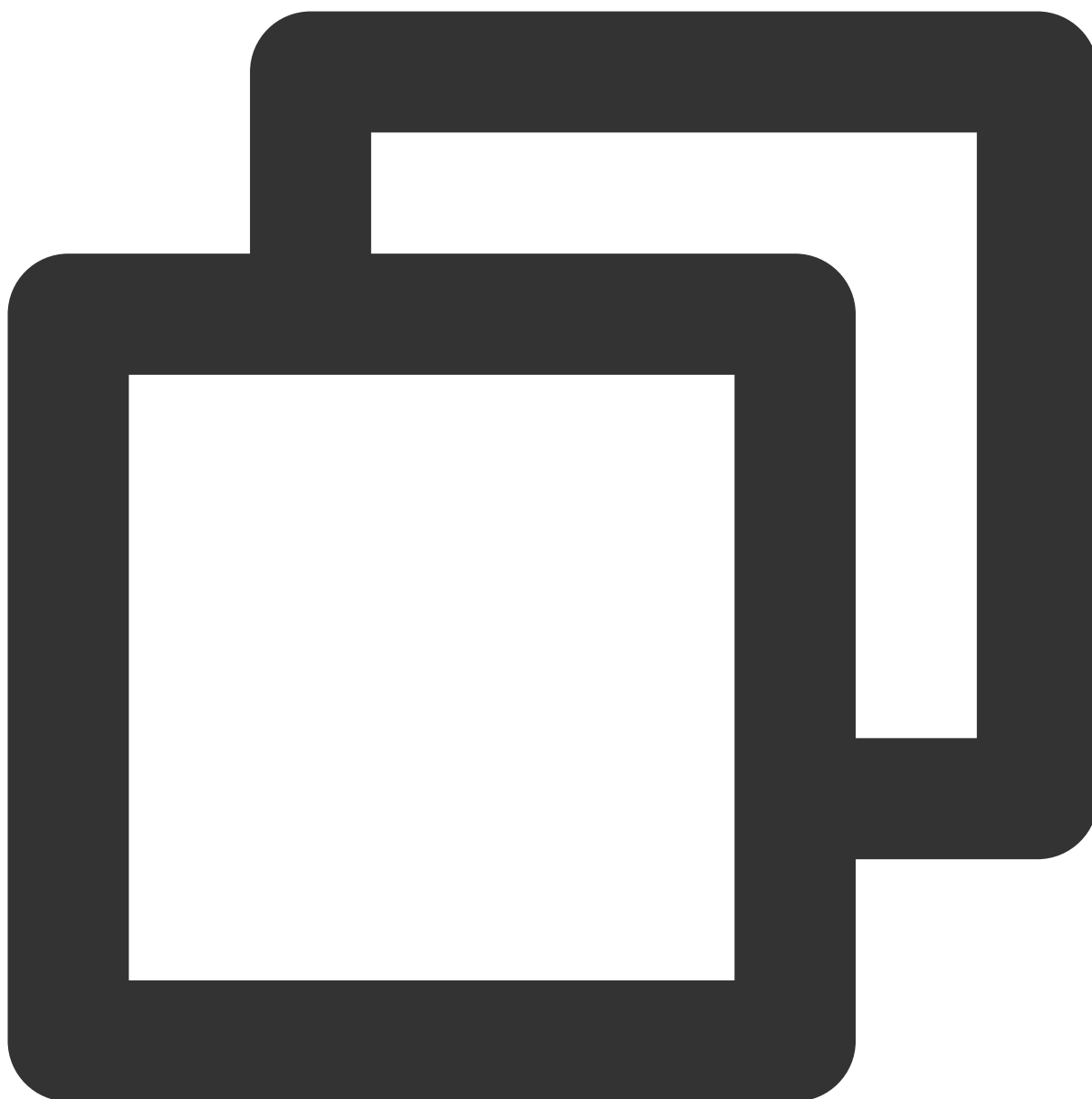
截屏、录屏事件和添加水印



```
- (void)applet:(TMFMiniAppInfo *)appletInfo screenCaptureStatusChanged:(BOOL)isCapt  
- (void)appletDidTakeScreenshot:(TMFMiniAppInfo *)appletInfo atPagePath:(NSString *)  
- (nullable UIView *)appletCustomizeWatermarkView:(TMFMiniAppInfo *)appletInfo;
```

web-view组件中特殊链接处理

对于 web-view 组件页面中要处理的特殊链接透传至宿主 app，由宿主 app 进行处理。



```
// web-view组件中跳转非http/https时触发，用于处理web-view组件中特殊url的情况，比如打开其它App
// @param app {TMFMiniAppInfo} web-view组件所在的小程序信息 - Mini-program information
// @param url {NSURL} 要打开的url - The url to open
// @return 是否拦截处理 - Whether to intercept processing
- (BOOL)webViewCustomUrlLoading:(TMFMiniAppInfo *)app url:(NSURL *)url {
    NSLog(@"webViewCustomUrlLoading:%@",appid:%@",[url absoluteString],app.appId);
    if ([[UIApplication sharedApplication] canOpenURL:url]) {
        if (@available(iOS 10.0, *)) {
            [[UIApplication sharedApplication] openURL:url options:@{ } completionHandler:
            if (success) {
                NSLog(@"webViewCustomUrlLoading:%@",appid:%@", open sucess",[url
```

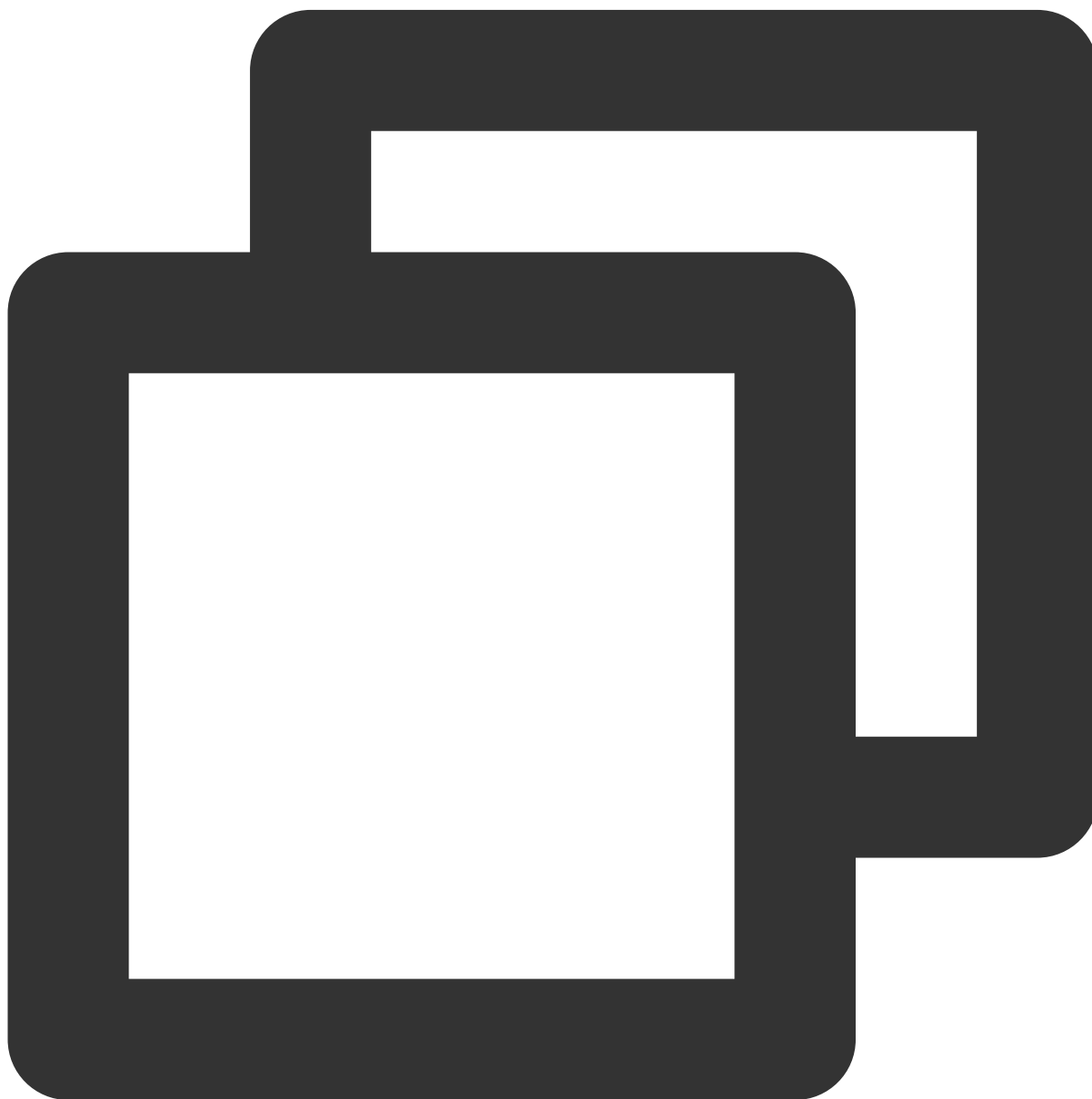
```
        }
    }];
} else {
    [[UIApplication sharedApplication] openURL:url];
}
return YES;
} else {
    NSLog(@"webViewCustomUrlLoading:%@,appid:%@,cann't open!!!",[url absoluteSt
    ]
    return NO;
}
```

API 说明

最近更新时间：2024-07-03 18:15:12

TMAVersionType

小程序版本类型

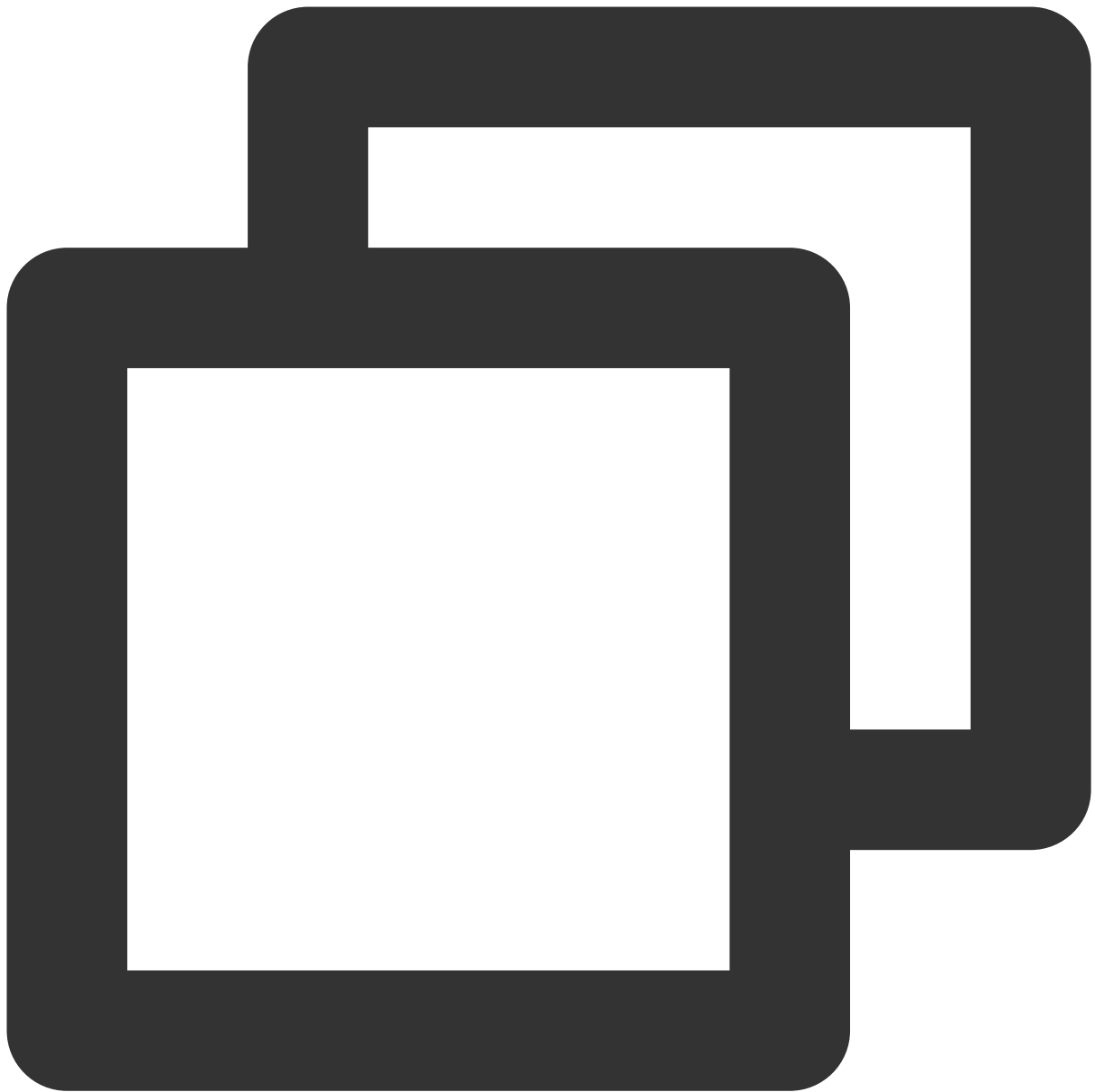


```
// 开发版本  
// Development version
```

```
TMAVersionDevelop = 0,  
  
// 预览版本  
// Preview version  
TMAVersionPreview = 1,  
  
// 审核版本  
// Audit version  
TMAVersionAudit = 2,  
  
// 线上版本（发布版本）  
// Online version (release version)  
TMAVersionOnline = 3,  
  
// 本地预置版本，不会进行更新（暂未使用）  
// Local preset version will not be updated (not used yet)  
TMAVersionLocal = 10,
```

TMAEntryScene

小程序打开的场景。



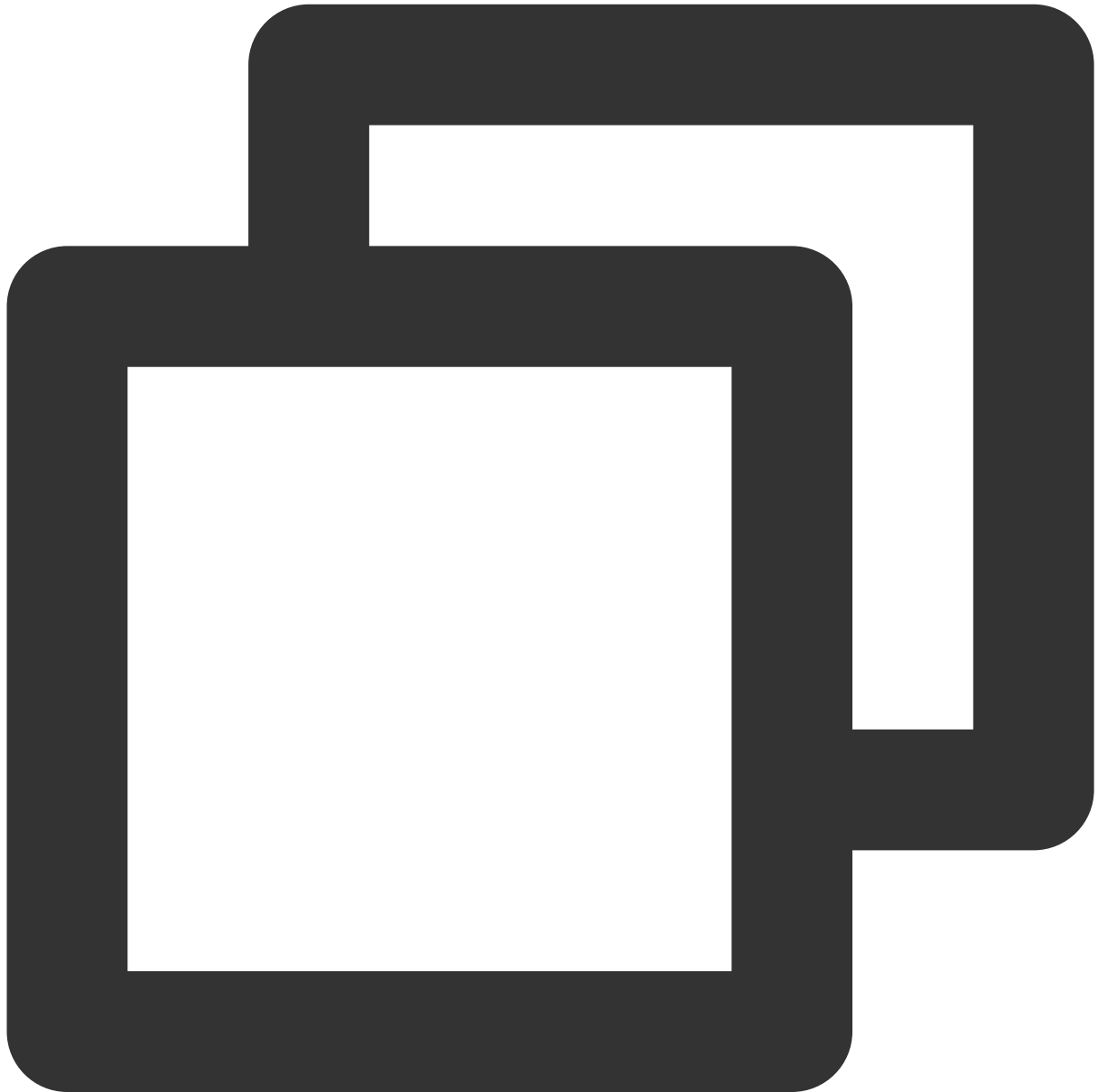
```
// 初始化值
// initialization value
TMAEntrySceneNone = 0,

// 主入口
// main entrance
TMAEntrySceneAIOEntry = 1001,

// 搜索
// search
```

```
TMAEntrySceneSearch = 1005,  
  
// 扫二维码  
//Scan QR code  
TMAEntrySceneScanQRCode = 1011,
```

MAShareTarget



```
// 分享到QQ
```

```
// Share to QQ
MShareTargetQQ = 0,

// 分享到Qzone
// Share to Qzone
MShareTargetQzone,

// 分享到QQ, 快速分享到当前聊天窗口
// Share to QQ, quickly share to the current chat window
MShareTargetQQFastForward,

// 微信好友
// WeChat friends
MShareTargetWXFriends,

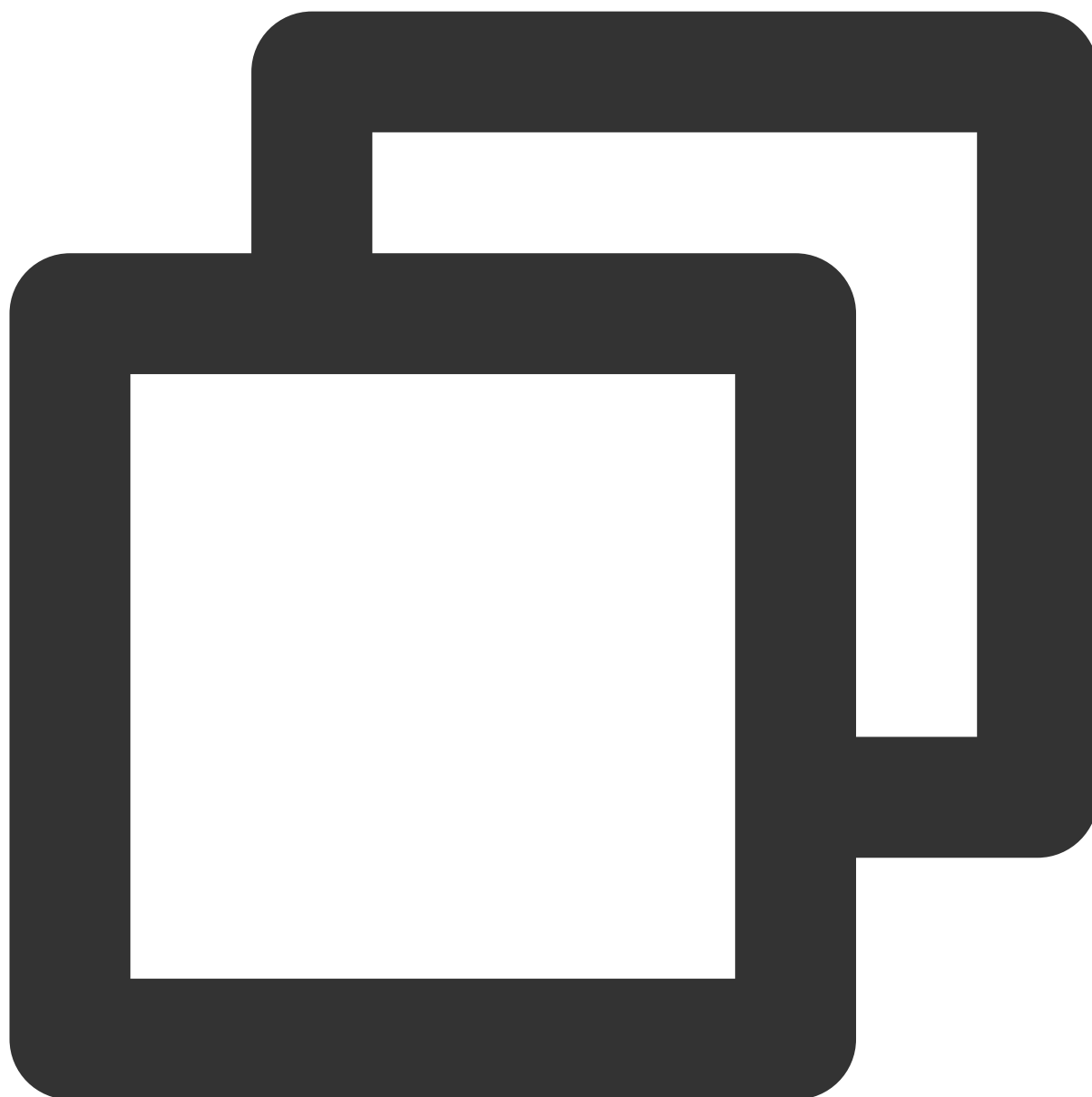
// 微信朋友圈
// WeChat Moments
MShareTargetWXMoment,

// 最近联系人快速分享, 从分享面板点击触发
// Quick sharing of recent contacts, triggered by clicking on the sharing panel
MShareTargetChatFastShare,

// 快捷分享好友列表
// Quickly share friends list
MShareTargetFastShareTable,

// openId分享, 来源小游戏的开放数据域点后后分享
// openId sharing, click and share the open data domain of the source mini game
MShareTargetOpenIdShare,
MShareTargetGuild,
```

TMAAppLifeCycleType



```
TMAAppLifeCycleTypeNone,  
  
TMAAppLifeCycleTypeonColdOpen,  
  
TMAAppLifeCycleTypeonHotOpen,  
  
TMAAppLifeCycleTypeOnShow,  
  
TMAAppLifeCycleTypeOnHide,
```

```
TMAAppLifeCycleTypeOnClose,  
  
TMAAppLifeCycleTypeTerminate,
```

扩展组件 SDK

最近更新时间：2024-07-24 10:04:47

在开发、使用中，很多需要系统授权的 API 需要在项目的 `info.plist` 文件中进行预置授权，但您的 App 可能并不需要此功能，所以 TCMPP SDK 拆分除了扩展 SDK，这样免去了不必要的授权，也缩小了核心模块的体积。

TCMPPSDK 引擎提供核心模块及扩展模块，方便使用者根据自己的情况进行接入。

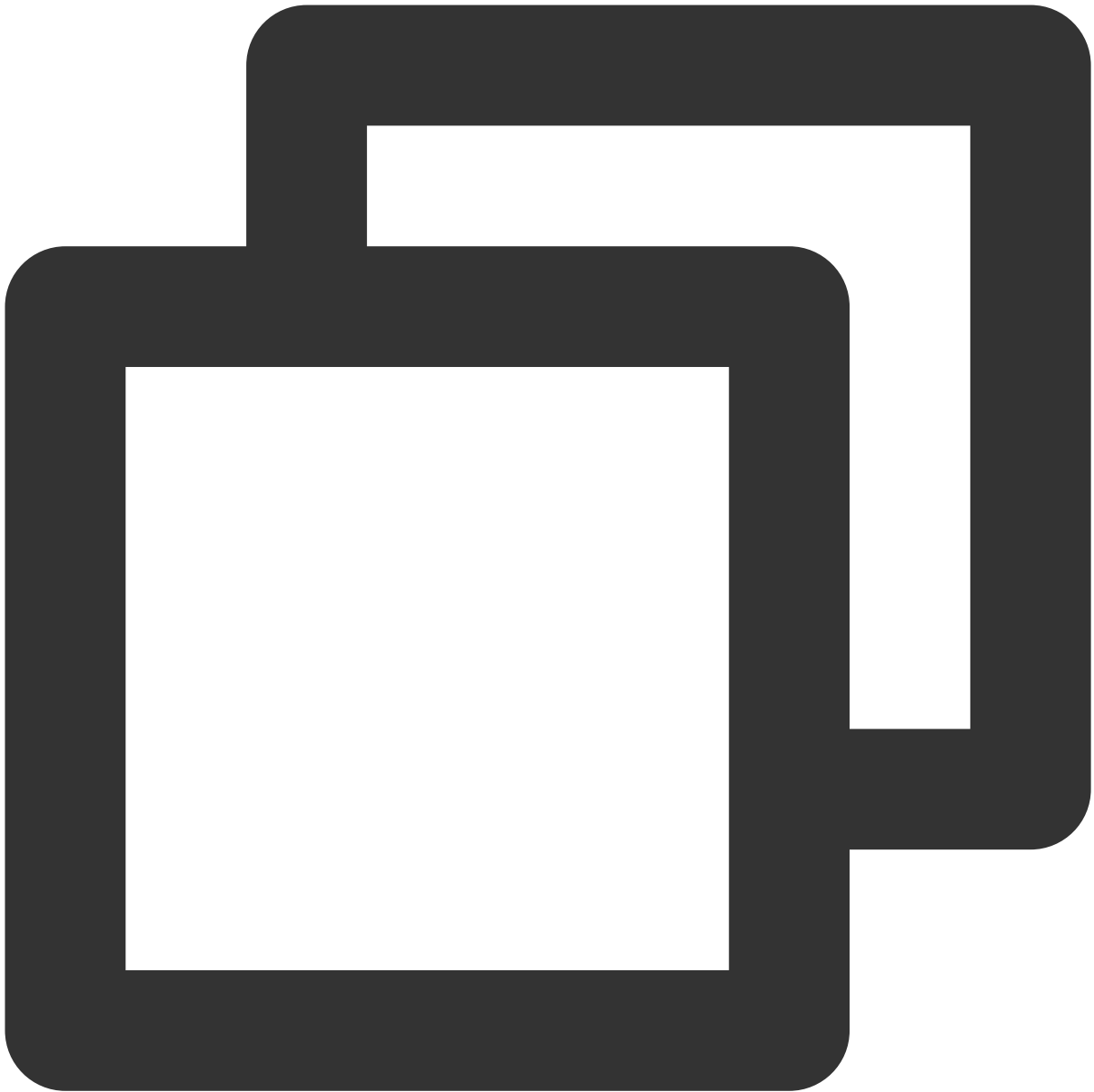
扩展 SDK 接入及使用

扩展 SDK 是对核心 SDK 的补充，所以要使用扩展 SDK，也必须依赖核心 SDK。为了保证 SDK 的安全稳定性，将需要权限的 API 尽可能放到扩展 SDK，TCMPPSDK 引擎将 SDK 拆分为核心 SDK 与扩展 SDK，后者是前者的补充，因此使用扩展 SDK 也必须依赖核心 SDK。

TCMPPExtMedia

TCMPPSDKExtMedia 提供 `chooseMedia`，`chooseVideo`，`chooseImage` 三个接口的默认实现，如果宿主 App 已经有对应能力，建议在开放接口中实现，如果需要使用 TMF 提供的多媒体选择插件，需要使用该插件。

使用方式：



```
pod 'TCMPPSDK'
pod 'TCMPPSDKExtMedia'
```

在 info.plist 文件中添加： Privacy - Photo Library Usage Description

添加 Media 扩展 SDK 后，增加支持的小程序 API 列表如下：

API 名称	说明
wx.chooseMedia	选择照片或视频
wx.chooseVideo	选择视频

wx.chooseImage	选择照片
----------------	------

涉及权限：

权限	描述
图库访问权限	需要申请访问图库的权限

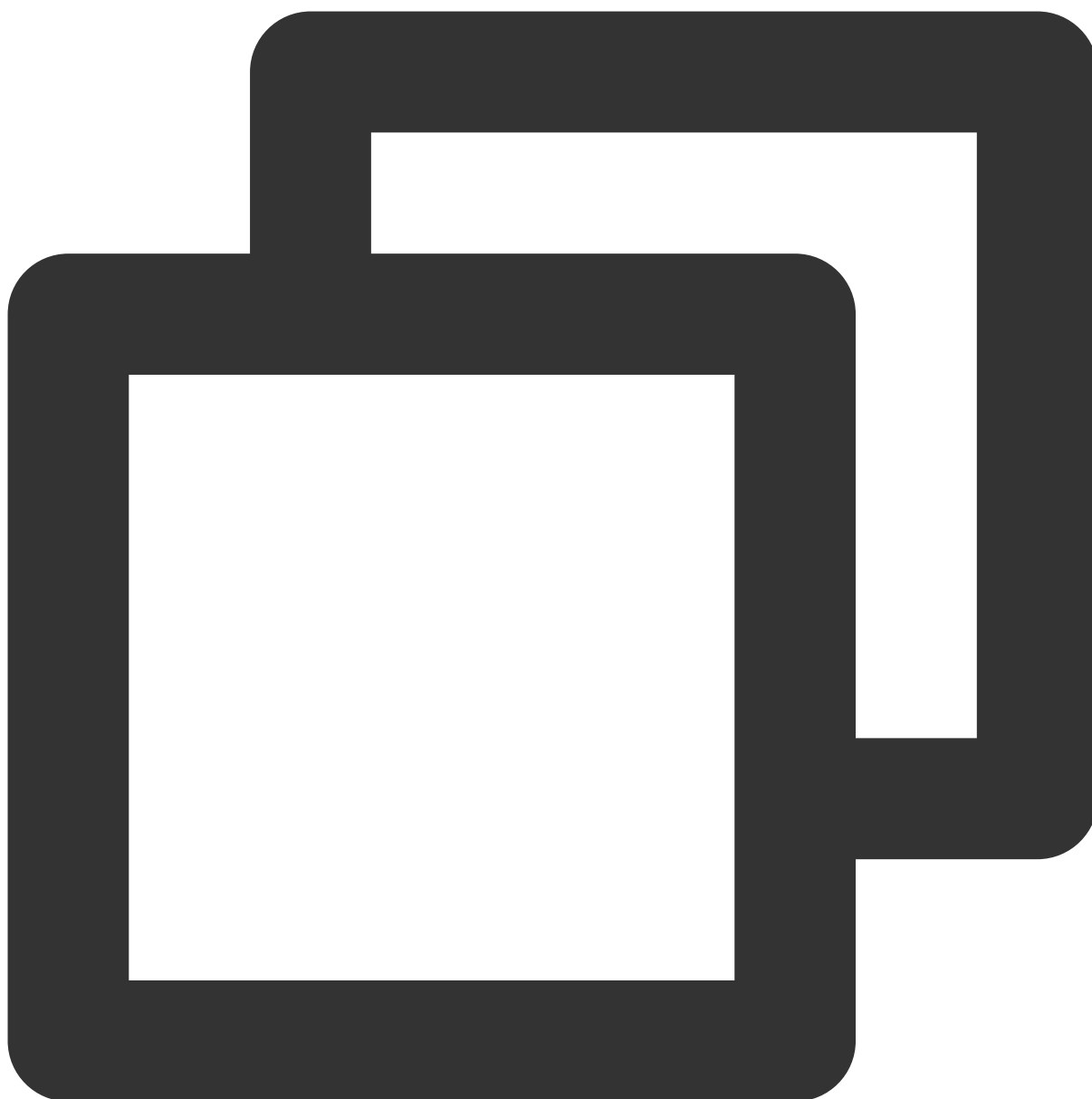
TCMPPExtScanCode

TCMPPSDKExtScanCode 提供 wx.scanCode 的处理逻辑，如果宿主 App 本身已经有扫码识别能力，建议通过。

```
TMFMiniAppSDKDelegate.scanCode:(NSDictionary *)scanPrams navigationController:
(UIINavigationController *)navigationController completionHandler:
(MACommonCallback) completionHandler;
```

对接已经正常使用的扫码模块，如果需要使用 TCMPP 提供的扫码功能，可以使用该插件。

使用方式：



```
pod 'TCMPPSDK'  
pod 'TCMPPSDKExtScanCode'
```

在 `info.plist` 文件中添加： `Privacy - Camera Usage Description`

添加扫码扩展 SDK 后，增加支持的小程序 API 列表如下：

API 名称	说明
<code>wx.scanCode</code>	调起客户端扫码界面进行扫码

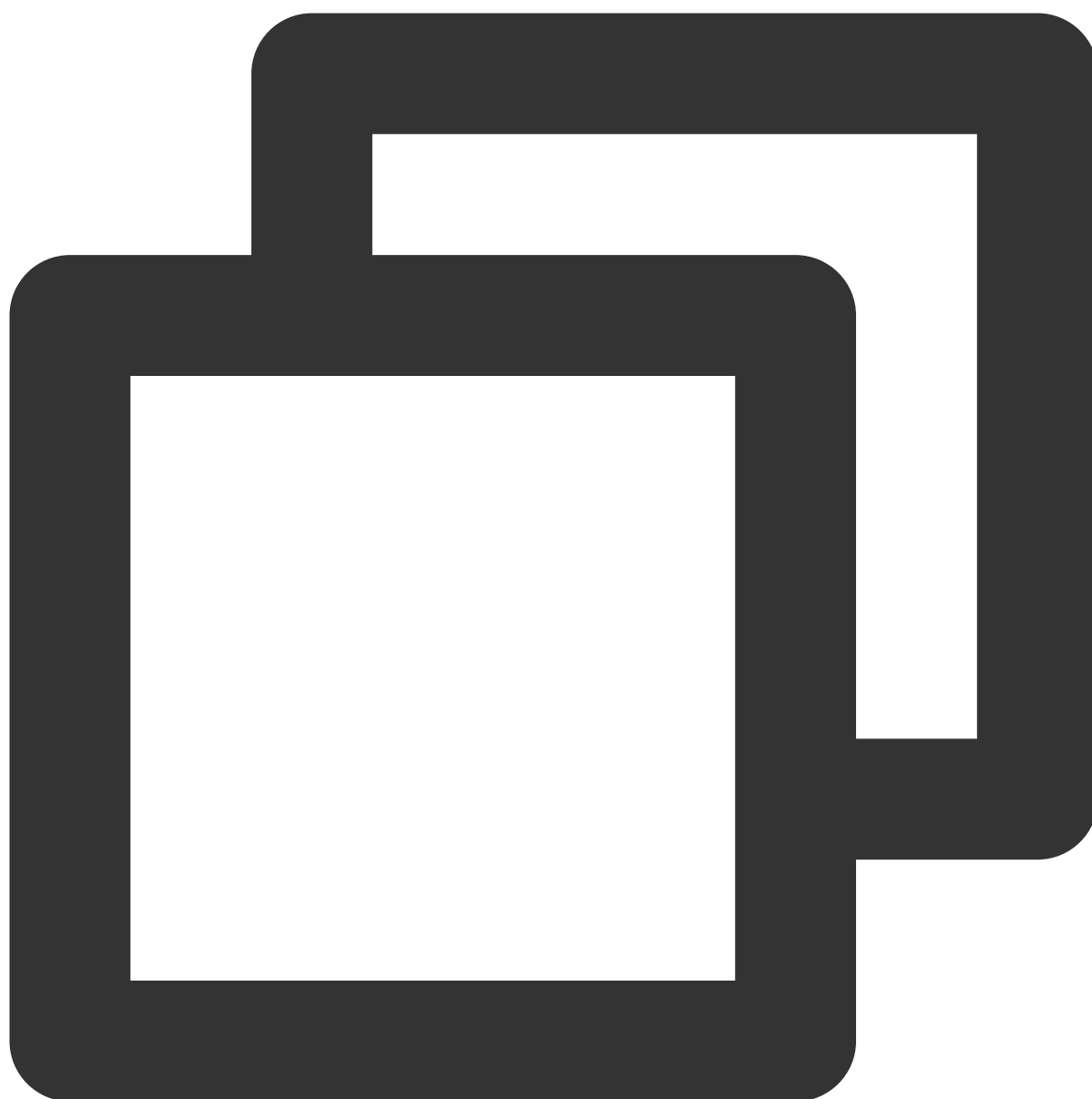
涉及权限：

权限	描述
相机权限	需要申请相机权限用于扫码
图库访问权限	需要申请访问图库的权限

TCMPPExtQMap

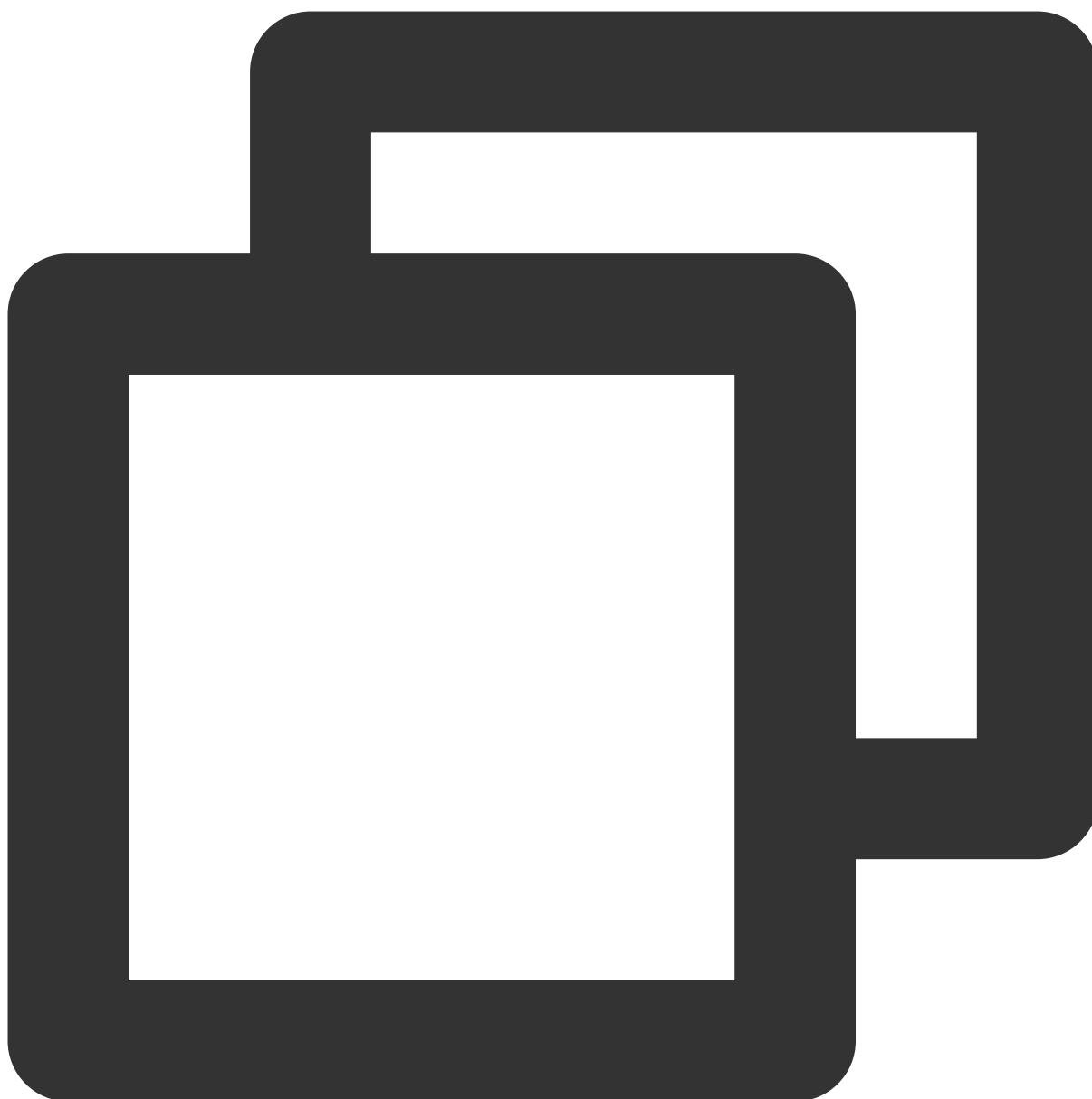
TCMPPSDKExtQMap 为中国大陆地区的用户提供腾讯地图 SDK 的相关能力。

使用方式：



```
pod "TCMPPSDKExtQMap"
```

在腾讯地图开放平台申请 **appkey** 并在代码中进行设置：

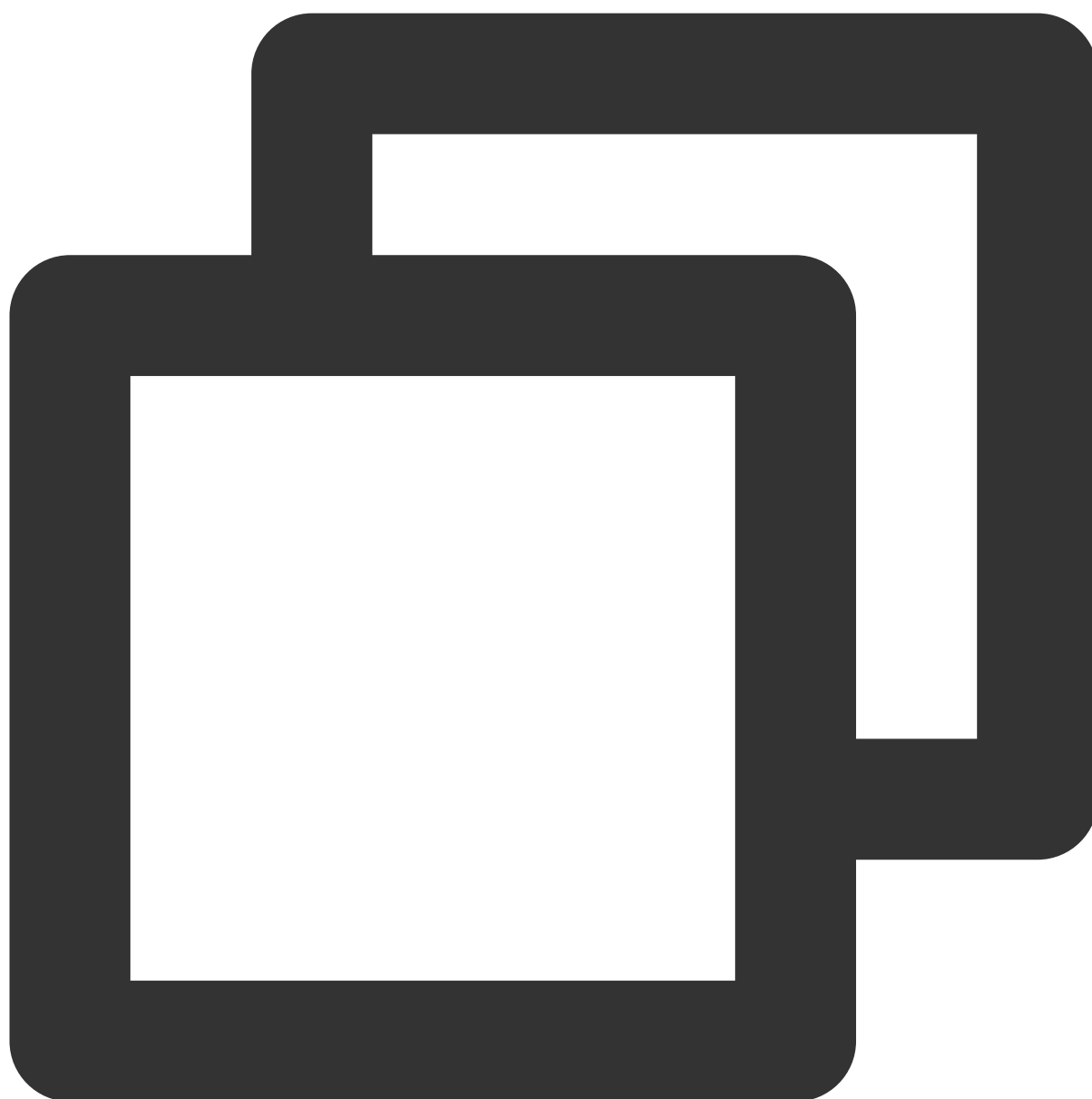


```
[TMFAppletQMapComponent setQMapApiKey:@"XXXXXXXXXXXX"];
```

TCMPPExtBaiduMap

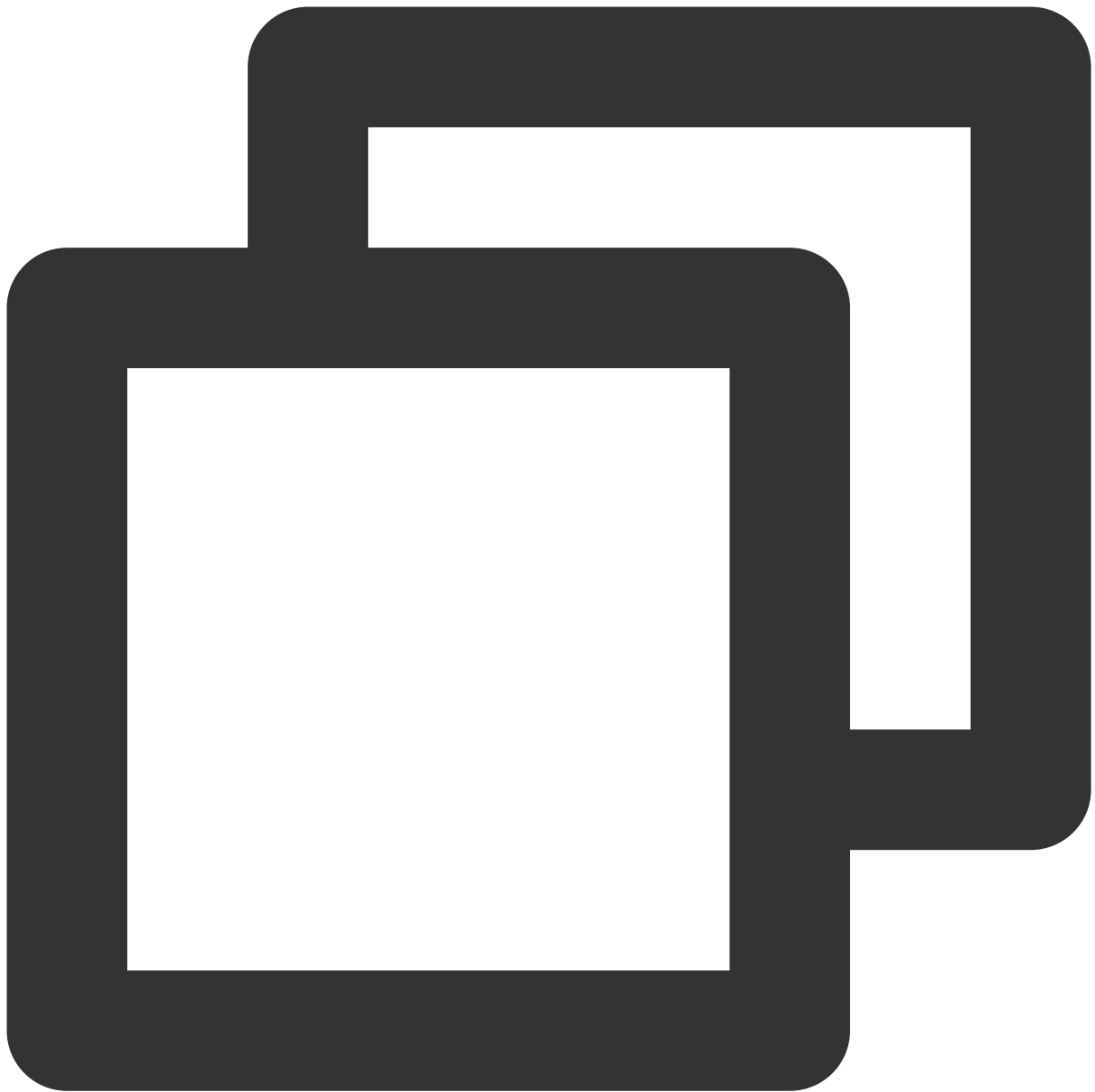
TCMPPSDKExtBaiduMap 为中国大陆地区的用户提供百度地图 SDK 的相关能力。

使用方式：



```
pod "TCMPPExtBaiduMap"
```

在百度地图开放平台申请 **appkey** 并在代码中进行设置：

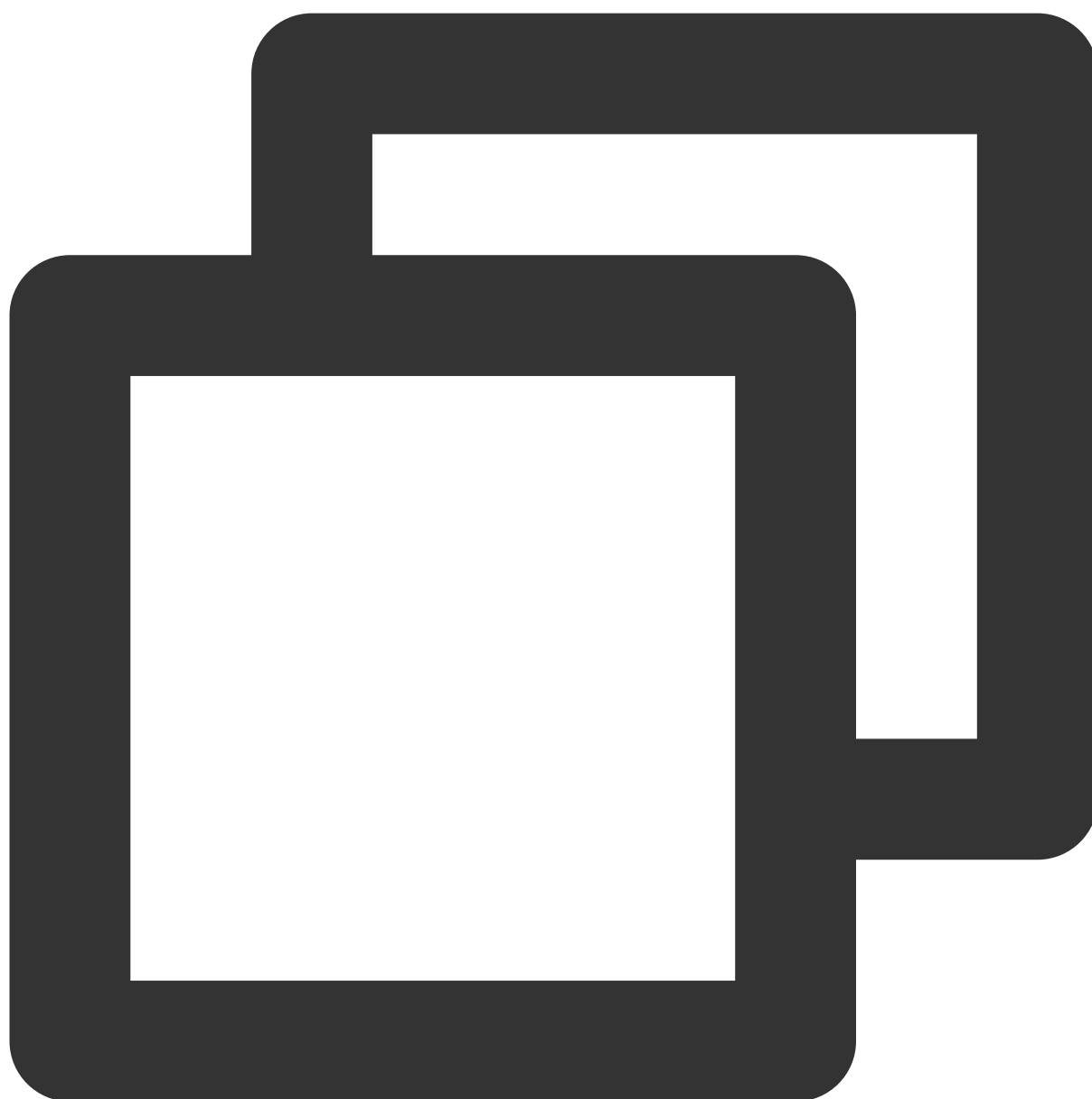


```
[TMFAppletBaiduMapComponent setBaiduMapApiKey:@"XXXXXXXXXXXXX"];
```

TCMPPExtAMap

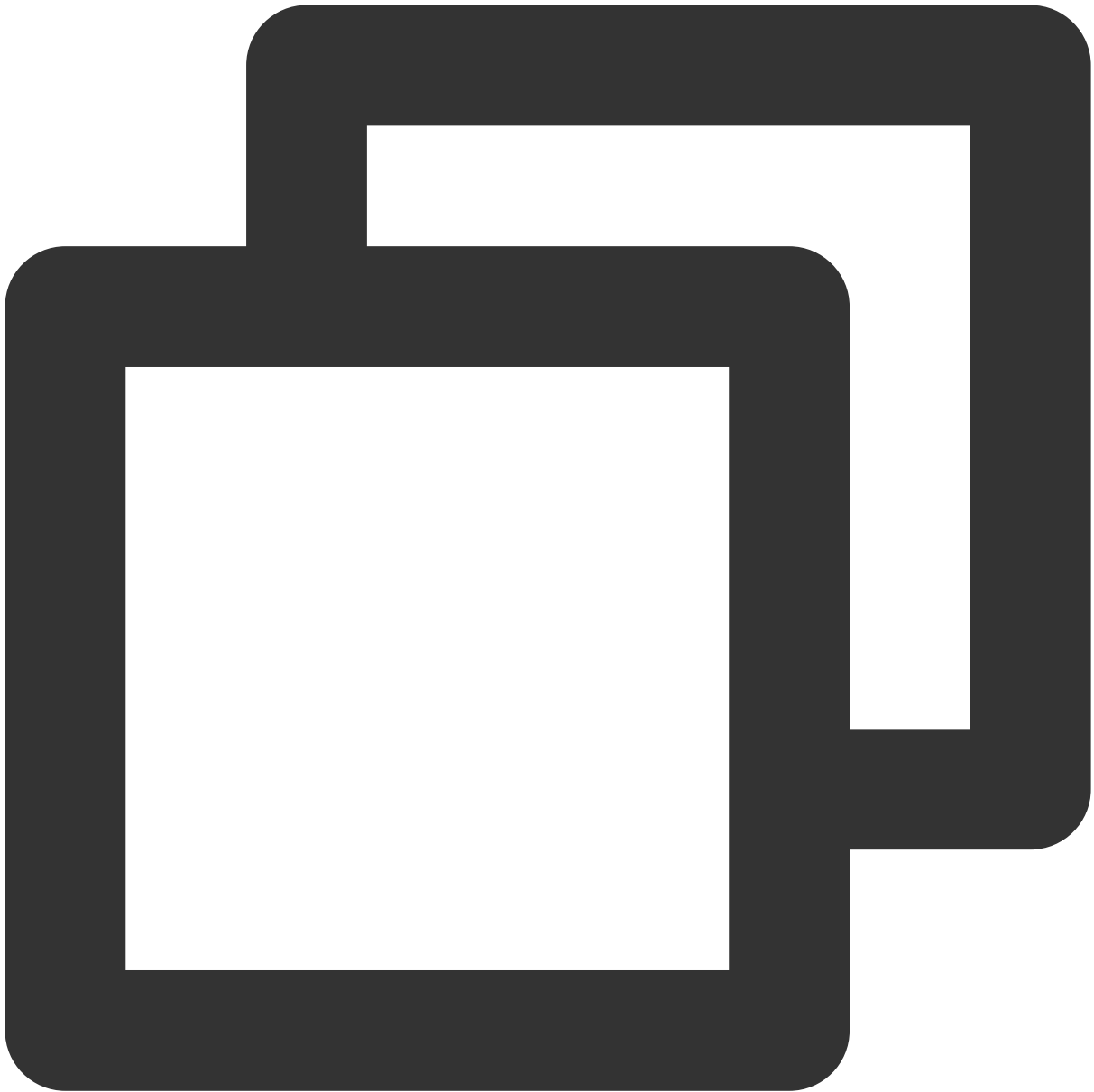
TCMPPSDKExtBaiduMap 为中国大陆地区的用户提供高德地图 SDK 的相关能力。

使用方式：



```
pod "TCMPPExtAMap"
```

在高德地图开放平台申请 **appkey** 并在代码中进行设置：



```
[TMFAppletAMapComponent setAMapApiKey:@"XXXXXXXXXXXXX"];
```

添加腾讯、百度、高德地图扩展 SDK 后，增加支持的小程序 API 列表如下：

API 名称	说明
地图	支持地图相关接口，包括地图展示，使用地图选择位置以及查询 POI 等

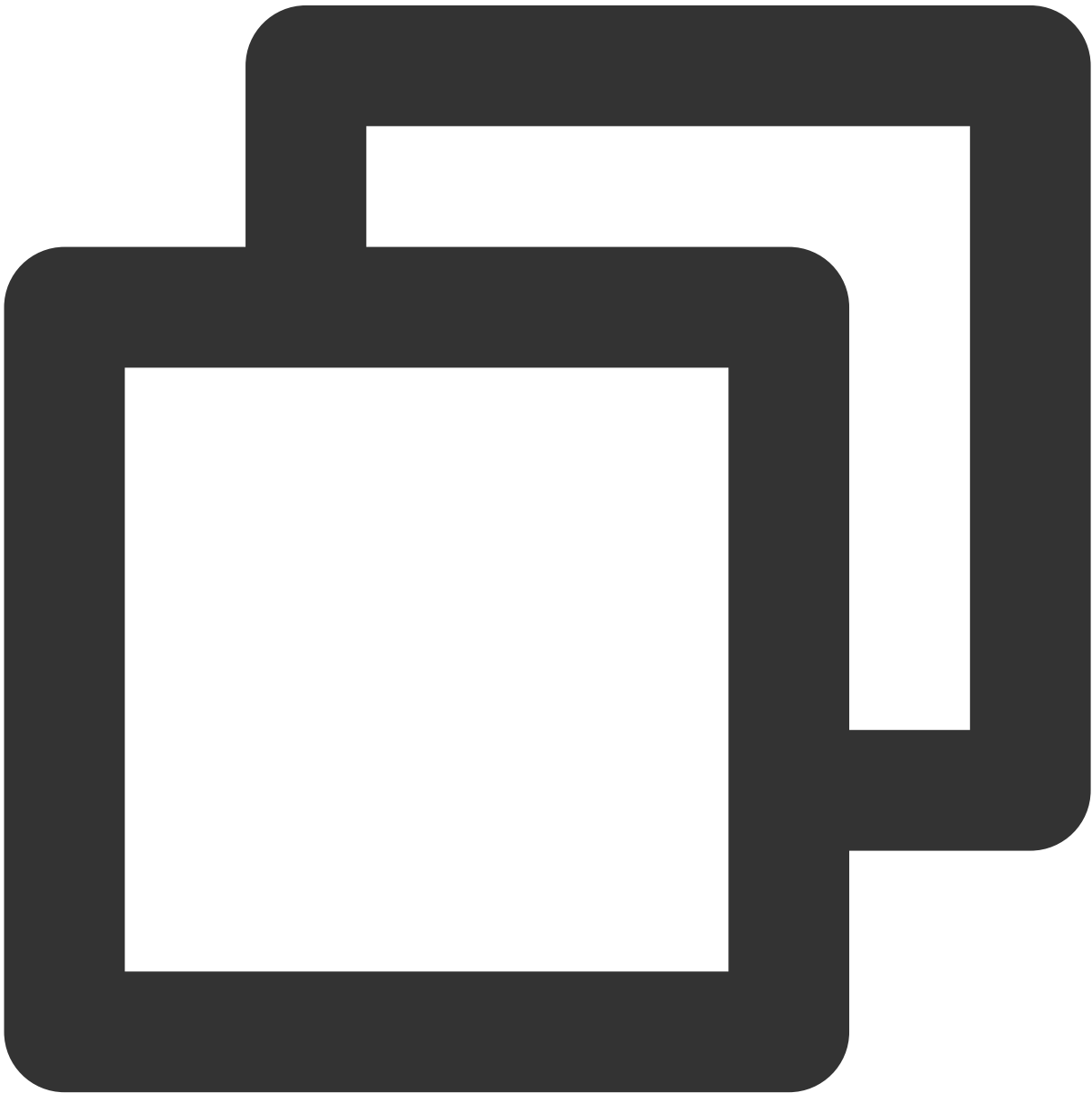
涉及权限：

--	--

权限	描述
定位权限	需要使用定位权限用于显示地图定位

TCMPPExtLive

如果您需要使用直播组件（live-player 和 live-pusher）进行直播推、拉流相关场景的开发，需要添加如下 SDK 以支持直播组件相关的功能的实现。

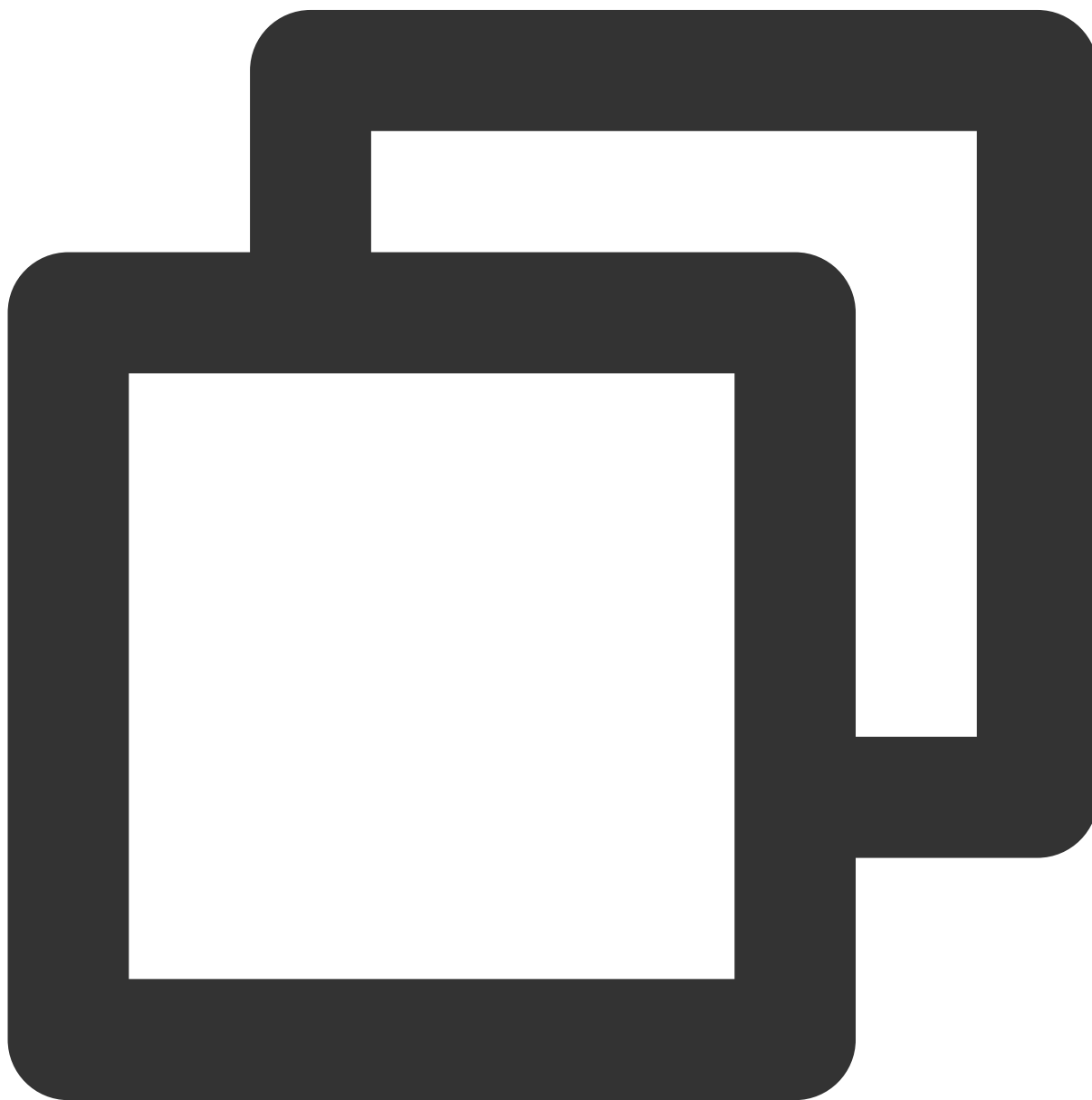


```
pod "TCMPPSDKExtLive"  
pod 'TXLiteAVSDK_Professional', :podspec => 'https://liteav.sdk.qcloud.com/pod/lite
```

除了完成以上依赖的添加，您还需要实现 `TMFMiniAppSDKDelegate` 中的如下方法，提供直播组件需要的 `LicenseUrl` 和 `LicenseKey`，以完成直播组件的初始化信息配置；如果您未配置正确的 `LicenseUrl` 和 `LicenseKey`，会导致直播组件功能不可用。

说明：

`LicenseUrl` 和 `LicenseKey` 的获取方式可以参考 [《新增与续期 License》](#)。



```
- (NSString *)setLiveLicenceURL {
```

```
        return @"https://xxx.license";
    }

    - (NSString *)setLiveLicenceKey {
        return @"xxx";
    }
```

在 info.plist 文件中添加：

```
Privacy - Camera Usage Description
Privacy - Microphone Usage Description
```

添加直播扩展 SDK 后，增加支持的小程序 API 列表如下：

API 名称	说明
wx.createLivePusherContext	创建直播推流端 context
LivePusherContext	支持 LivePusherContext 相关接口
wx.createLivePlayerContext	创建直播拉流端 context
LivePlayerContext	支持 LivePlayerContext 相关接口
组件	-
live-pusher	推流标签
live-player	播放标签

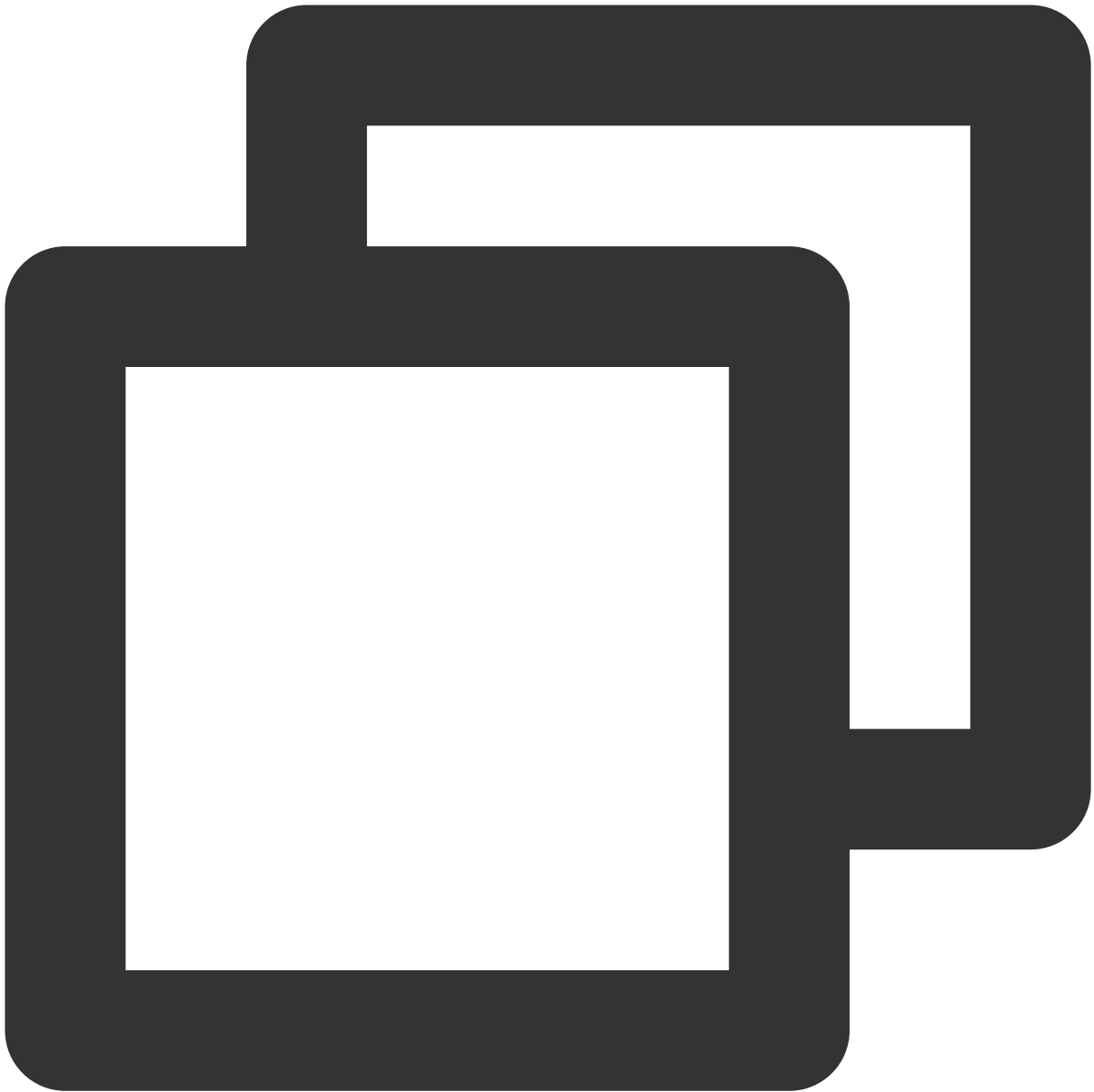
涉及权限如下：

权限名称	描述
相机权限	-
录音权限	-

TCMPPExtAuthentication

TCMPPExtAuthentication 提供生物认证相关的能力。

集成方式：



```
pod "TCMPPExtAuthentication"
```

使用说明：

在 info.plist 文件中添加： Privacy - Face ID Usage Description

添加生物认证扩展 SDK 后，增加支持的小程序 API 列表如下：

API 名称	说明
wx.startSoterAuthentication	-

wx.checkIsSupportSoterAuthentication	-
wx.checkIsSoterEnrolledInDevice	-

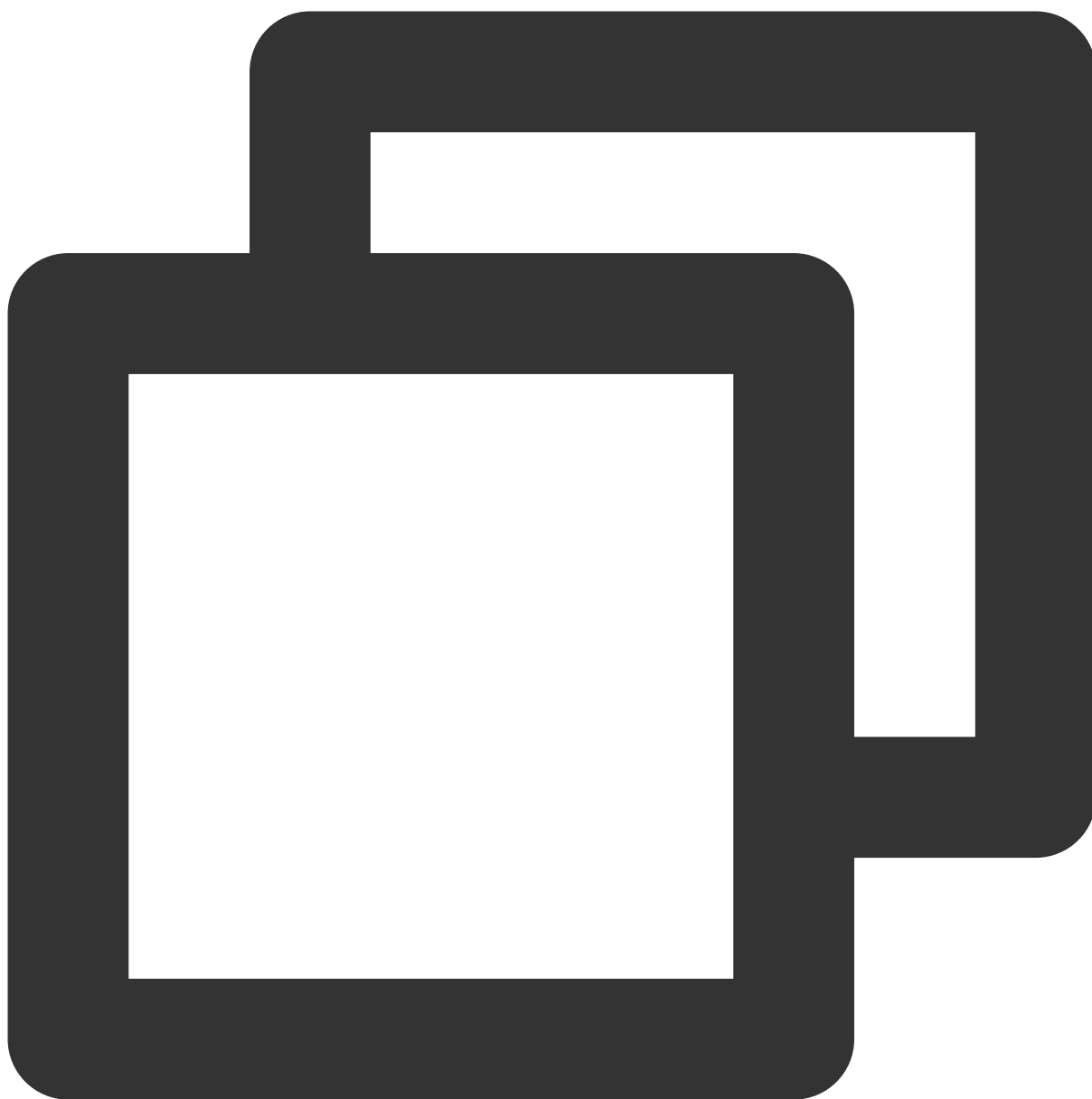
涉及权限：

权限	描述
生物认证访问	需要生物认证权限

TCMPPExtBLE

TCMPPExtBLE 提供低功耗蓝牙及信标相关的能力。

集成方式：



```
pod "TCMPPExtBLE"
```

使用说明：

在 info.plist 文件中添加：

```
Privacy - Bluetooth Always Usage Description
```

```
Privacy - Bluetooth Peripheral Usage Description
```

添加 LBS 扩展 SDK 后，增加支持的小程序 API 列表如下：

API	说明
-----	----

蓝牙-通用	蓝牙通用接口
蓝牙-低功耗外围设备	外围设备相关接口
蓝牙-低功耗中心设备	中心设备相关接口
蓝牙-信标	蓝牙信标相关接口

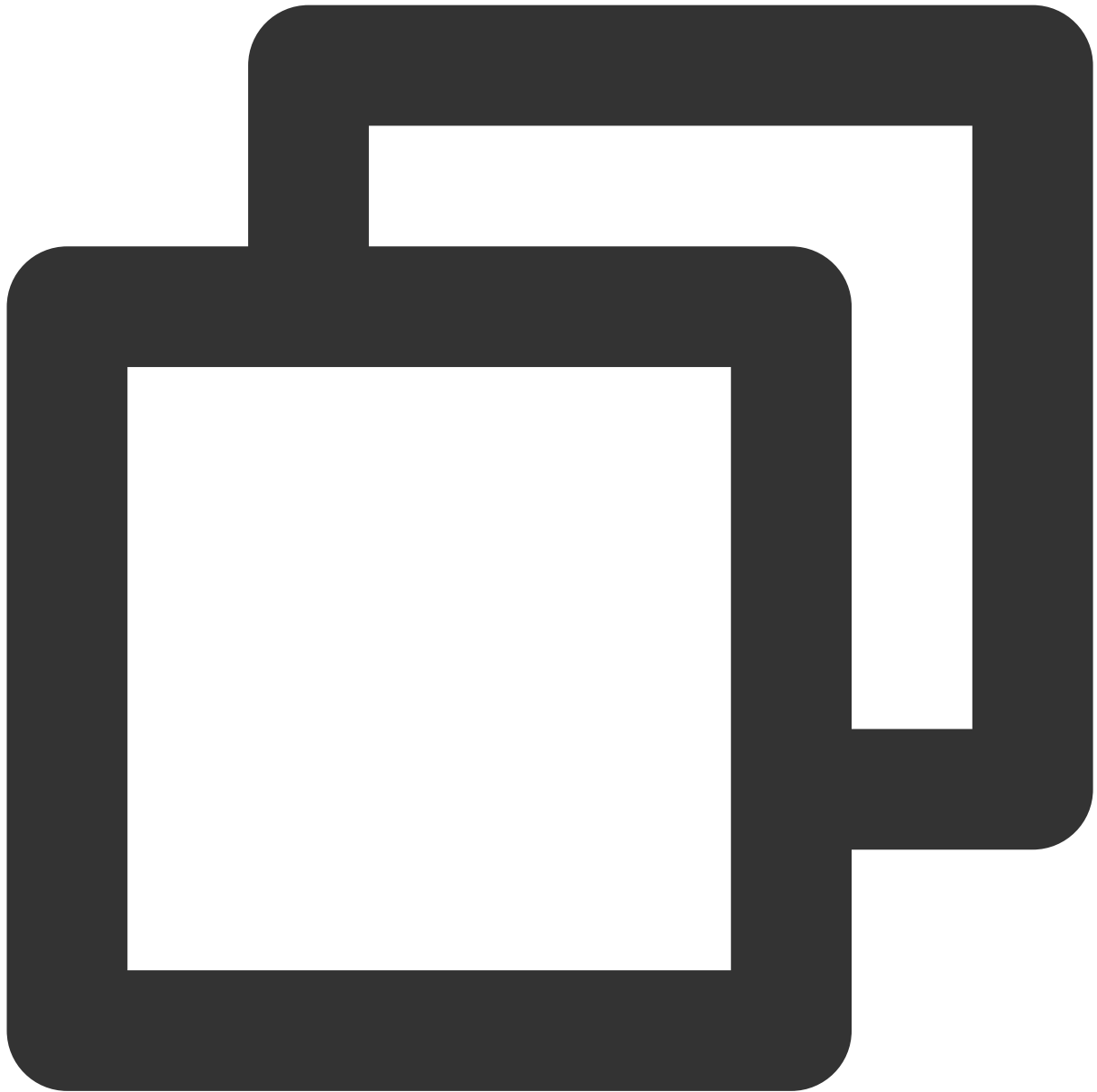
蓝牙扩展 SDK 涉及权限如下：

权限	说明
蓝牙	操作蓝牙需要申请蓝牙权限
定位	蓝牙设备搜索依赖定位权限

TCMPPExtCalendar

TCMPPExtCalendar 提供日历相关的能力。

集成方式：



```
pod "TCMPPExtCalendar"
```

使用说明：

在 info.plist 文件中添加：

```
Privacy - Calendars Usage Description
```

```
Privacy - Reminders Usage Description
```

API 列表：

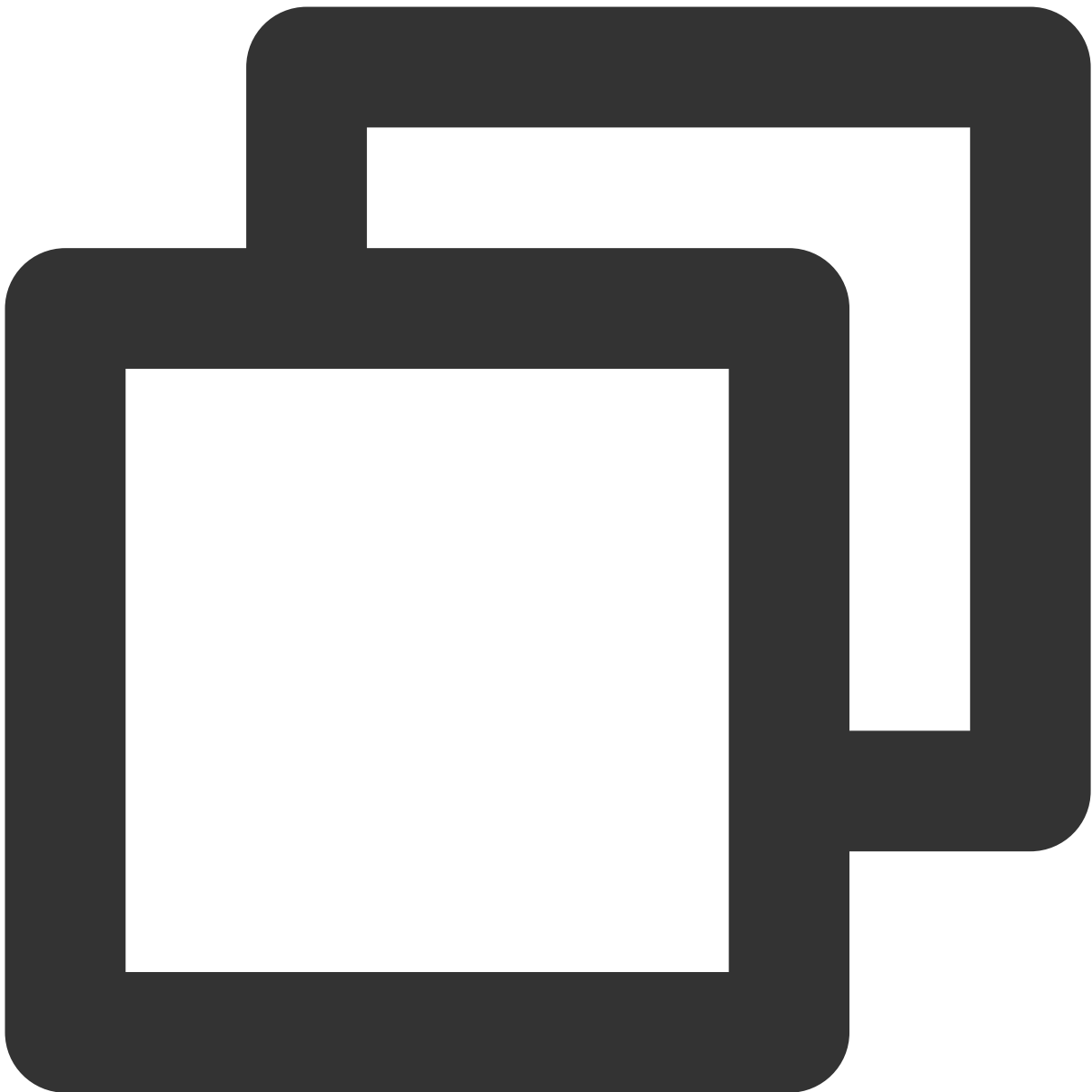
--	--

API 名称	API 描述信息
addPhoneRepeatCalendar	向系统日历添加重复事件
addPhoneCalendar	向系统日历添加事件

TCMPPExtClipBoard

TCMPPExtClipBoard 提供剪贴板相关的能力。

集成方式：



```
pod "TCMPPExtClipBoard"
```

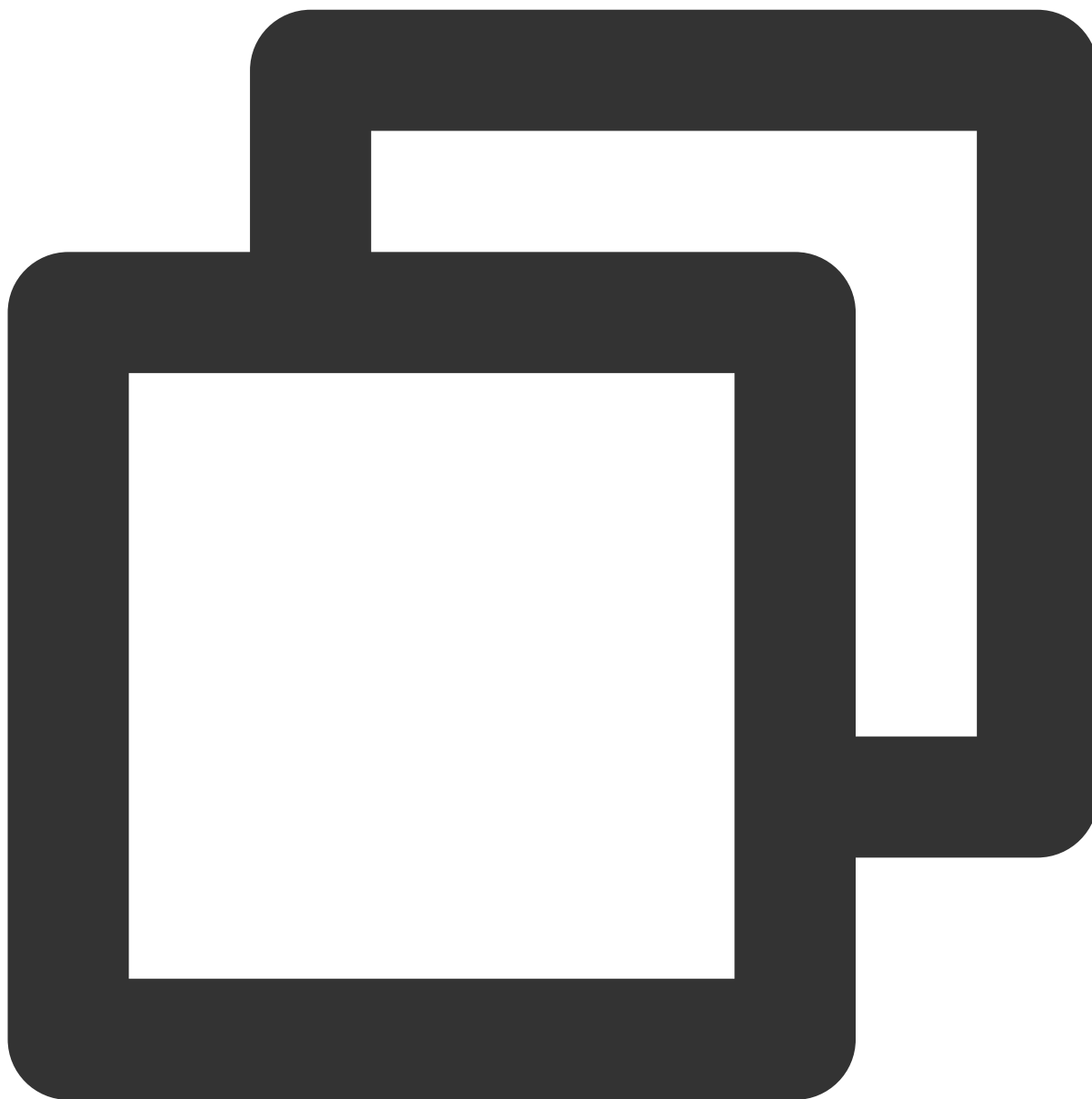
API 列表：

API 名称	API 描述信息
setClipboardData	设置系统剪贴板的内容
getClipboardData	获取系统剪贴板的内容

TCMPPExtContact

TCMPPExtContact 提供联系人相关的能力。

集成方式：



```
pod "TCMPPExtContact"
```

使用说明：

在 info.plist文件中添加： `Privacy - Contacts Usage Description`

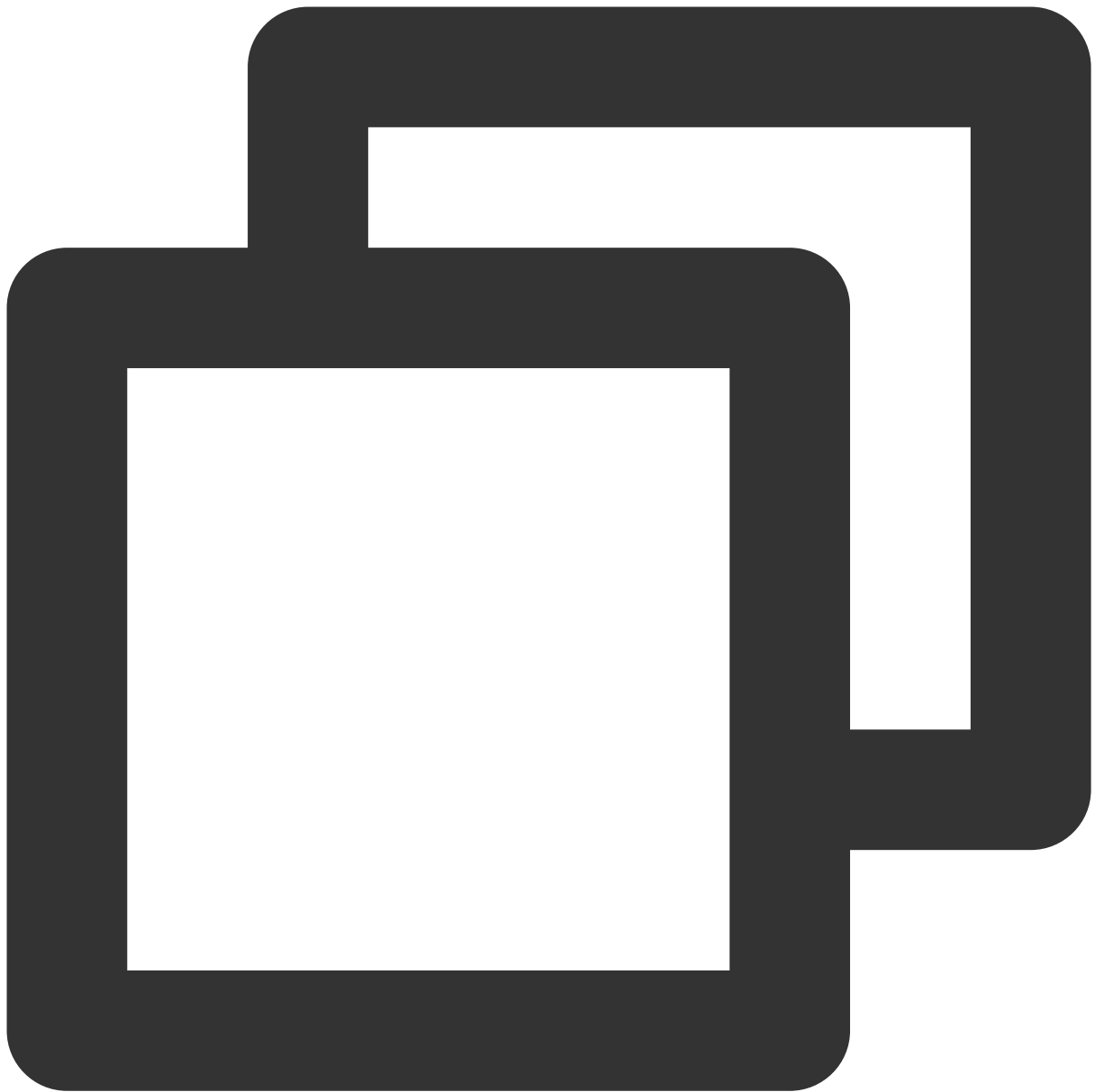
API 列表：

API 名称	API 描述信息
chooseContact	选择联系人
addPhoneContact	添加手机通讯录联系人

TCMPPExtLBS

TCMPPExtLBS 提供系统定位、系统地图、罗盘、加速计、设备方向、陀螺仪相关的能力。

集成方式：



```
pod "TCMPPExtLBS"
```

使用说明：

在 info.plist 文件中添加：

```
Privacy - Location Always and When In Use Usage Description
```

```
Privacy - Location Always Usage Description
```

```
Privacy - Location Usage Description
```

```
Privacy - Location When In Use Usage Description
```

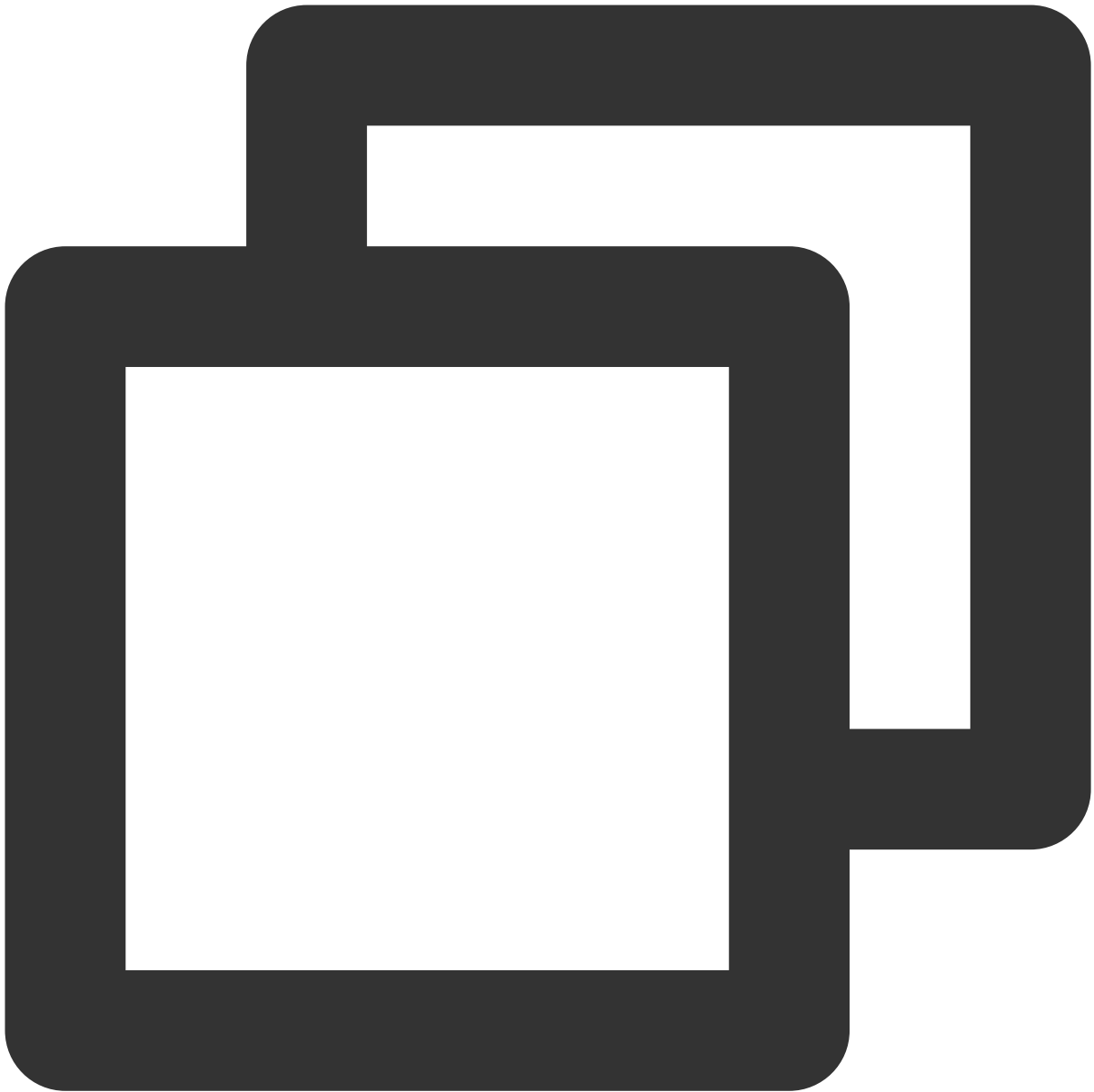
API 列表：

API 名称	API 描述信息
定位	一系列的api
系统地图	一系列的api
罗盘、加速计、设备方向、陀螺仪	一系列的api

TCMPPExtMDNS

TCMPPExtMDNS 提供局域网通信的能力。

集成方式：



```
pod "TCMPPExtMDNS"
```

使用说明：

在info.plist 文件中添加：

- Privacy - Local Network Usage Description
- Privacy - Bonjour services

API 列表：

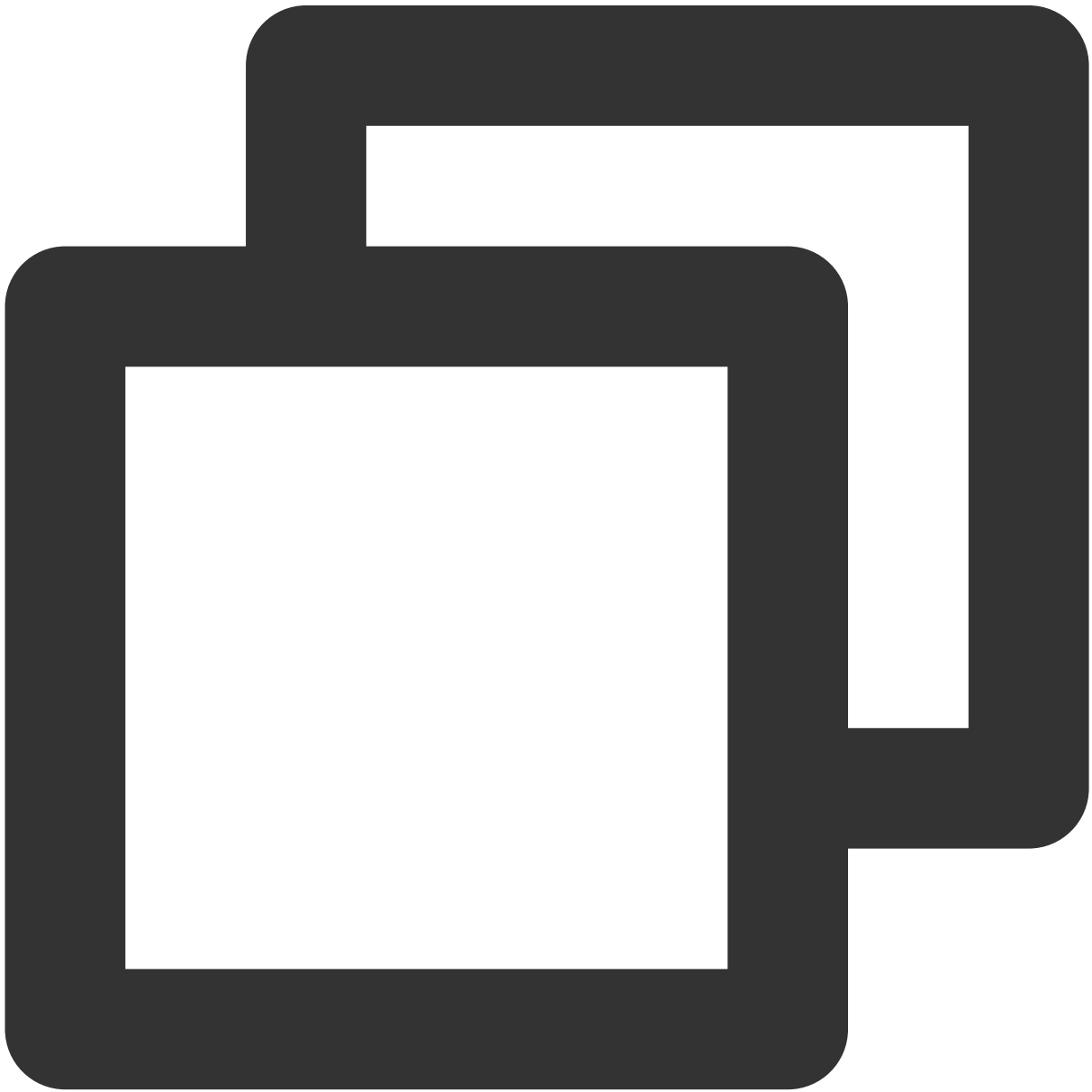
--	--

API 名称	API 描述信息
MDNS	一系列的 api

TCMPPExtNetwork

TCMPPExtNetwork 提供 TCP/UDP 通信的能力。

集成方式：



```
pod "TCMPPExtNetwork"
```

使用说明：

在 info.plist 文件中添加：

```
App Transport Security Settings
Allow Arbitrary Loads - YES
Privacy - Bonjour services
```

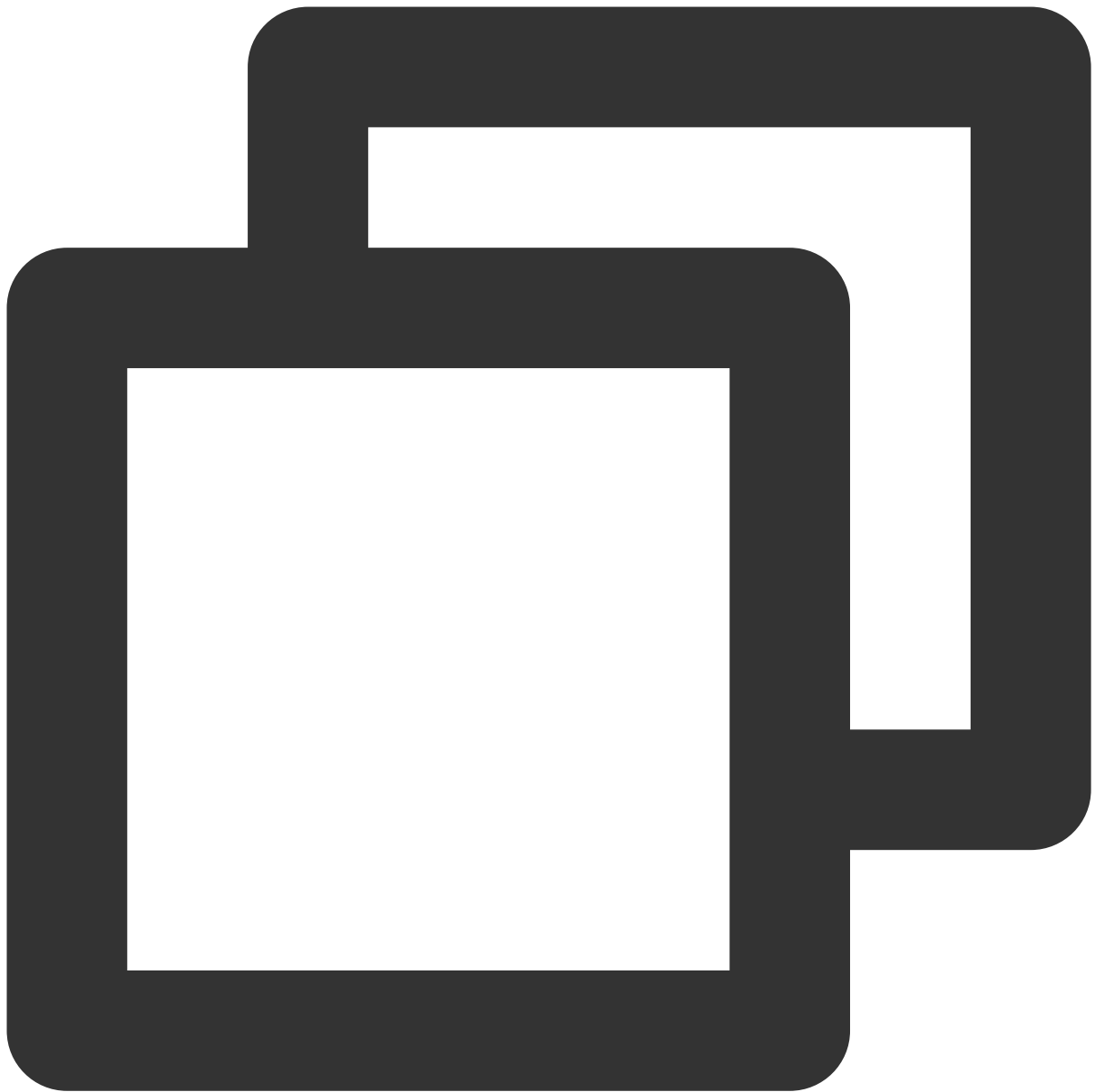
API 列表：

API名称	API 描述信息
TCP	一系列的 api
UDP	一系列的 api

TCMPPExtMp3Encoder

TCMPPExtMp3Encoder 提供使用 RecorderManager 时保存为 mp3 格式的能力。

集成方式：



```
pod "TCMPPExtMp3Encoder"
```

使用说明：

TCMPP 提供的录音保存成 mp3格式依赖 Lame 库，Lame 库开源协议基于 **GNU Library or Lesser General Public License version 2.0 (LGPLv2), GNU General Public License version 2.0 (GPLv2)** 参考：[LAME](#)，使用者可以根据需要集成。

iOS SDK 常见问题

最近更新时间：2024-07-29 16:48:42

集成时常见问题

集成 SDK 的最低配置是什么？

TCMPP 小程序 SDK 支持 iOS 9 以及 iOS 9 以上，低于 iOS 9 的系统版本不支持。

XCode 15 上编译后，在 ios12 及以下设备上出现崩溃问题

在项目编译选项中添加 `-Wl,-ld_classic` 配置。



Xcode 15 RC Release Notes

Linking



New Features

A new linker has been written to significantly speed up static linking. It's the default for all macOS, iOS, tvOS and visionOS binaries and using the "Mergeable Libraries" feature. The classic linker can still be explicitly requested using `-ld64`, and will be removed in a future version (108915312)

Known Issues

Binaries using symbols with a weak definition crash at runtime on iOS 14/macOS 12 or older. This impacts primarily C++ projects due to extensive use of weak symbols. (114813650) ([FB13097713](#))

Workaround: Bump the minimum deployment target to iOS 15, macOS 12, watchOS 8 or tvOS 15, or add `-Wl,-ld_classic` to the `OTHER_LDFLAGS` build setting.

集成 SDK 后对安装包的大小有什么影响

TCMPP SDK 使用核心库和扩展库的方式，方便使用者根据自己的需要来集成 SDK，在只集成核心库的情况下，对整体安装包的大小影响在 4M 左右。

使用时常见问题

小程序启动失败排查

小程序启动失败，可能有如下几种原因：

原因一：配置文件路径错误，如果配置文件在子目录下需要追加目录路径，例如：`server/tcmpp-ios-configurations.json`。

原因二：不允许修改小程序配置文件内容，否则小程序无法正常运行。

原因三：配置文件中的 `bundleId` 必须与应用的 `bundleId` 保持一致，否则 App 运行失败，因为初始化时会校验配置文件中的 `bundleId`。


```
{
  "customId": "T1701314211HMOVUJP",
  "productId": "4007",
  "packageName": "",
  "bundleId": "com.tencent.tcmpp.demo",
  "material":
    "01+cpJbB8WktGpiif0wnkta7qerkNi/MAGHHx9KF3GzPVX1tn/DpJHvuaO11
    bZBRhxdCopWC16RlU/zla2oa3PK3ayWibFCPjFL08RZJpHtnKweNMSJIO0cDy
    K5oQUR9fs2AARuY4kSIR4gLLb7ZnfjUVdViHAHPmxPr5IJG2Lt9bonLa8Zhbf
    elp2RdvEA=1fd53748f1db282e",
  "shark": {
    "httpUrl": "https://api.tcmpp.tmfcloud.com",
    "tcpHost": "api.tcmpp.tmfcloud.com",
    "tcpPort": 0,
    "appKey": "app-azosmbqqjn",
    "symEnc": 2,
    "asymEnc": 1
  },
  "feedback": "",
  "qapmUrl": "http://ww.com"
}
```

是否支持小程序某页面完全隐藏导航栏？如果支持，该如何实现？

答：支持。

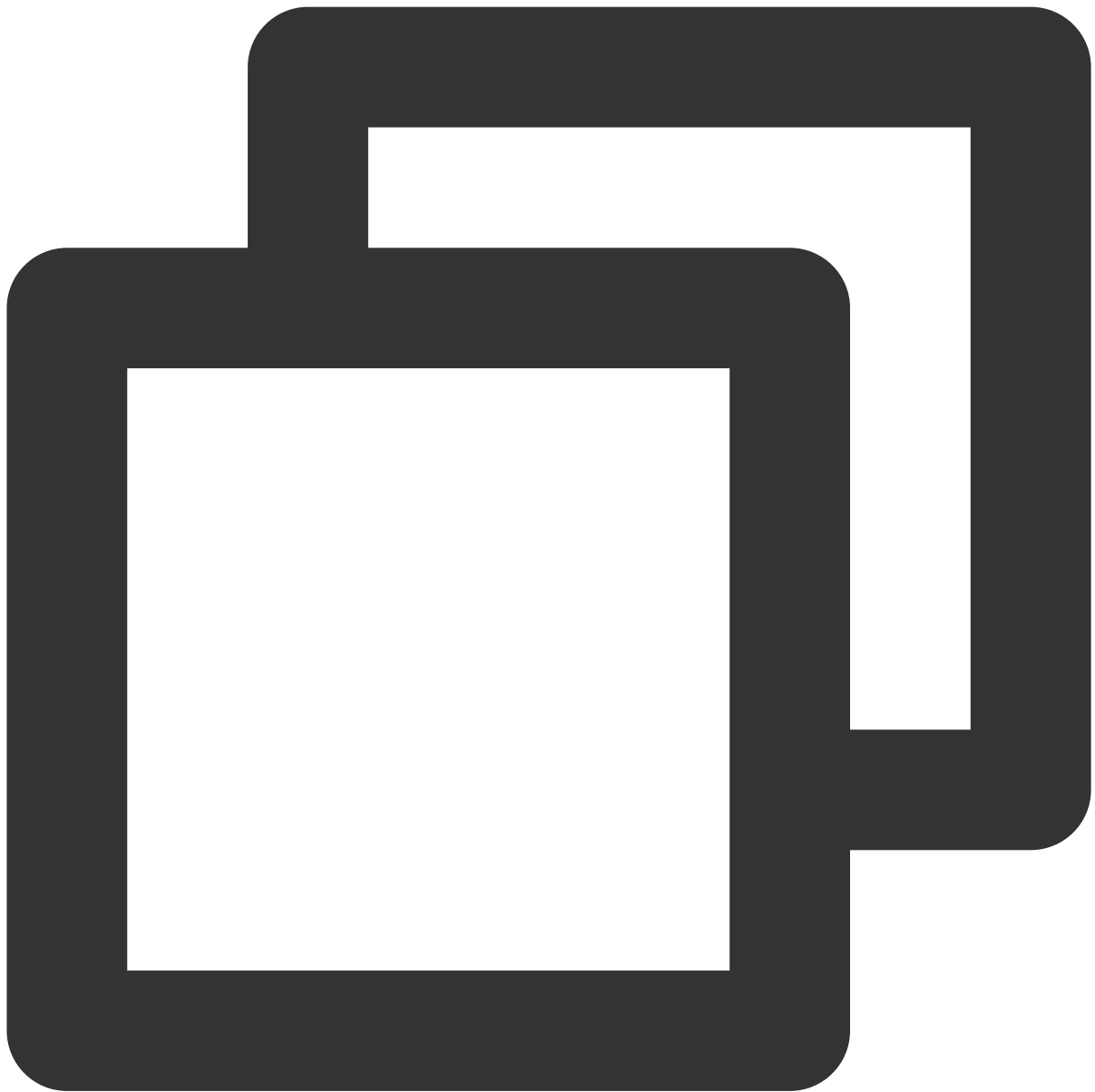
像微信、支付宝、百度等小程序的 `navigationStyle` 有 `default/custom` 两个值，而 TCMPP 小程序 SDK 在此基础上，新增了一个 `hide`，即有 `default/custom/hide` 三个值。

所以在需要隐藏导航栏的页面，设置 `navigationStyle` 为 `hide` 即可。

如何设置灰度发布条件？

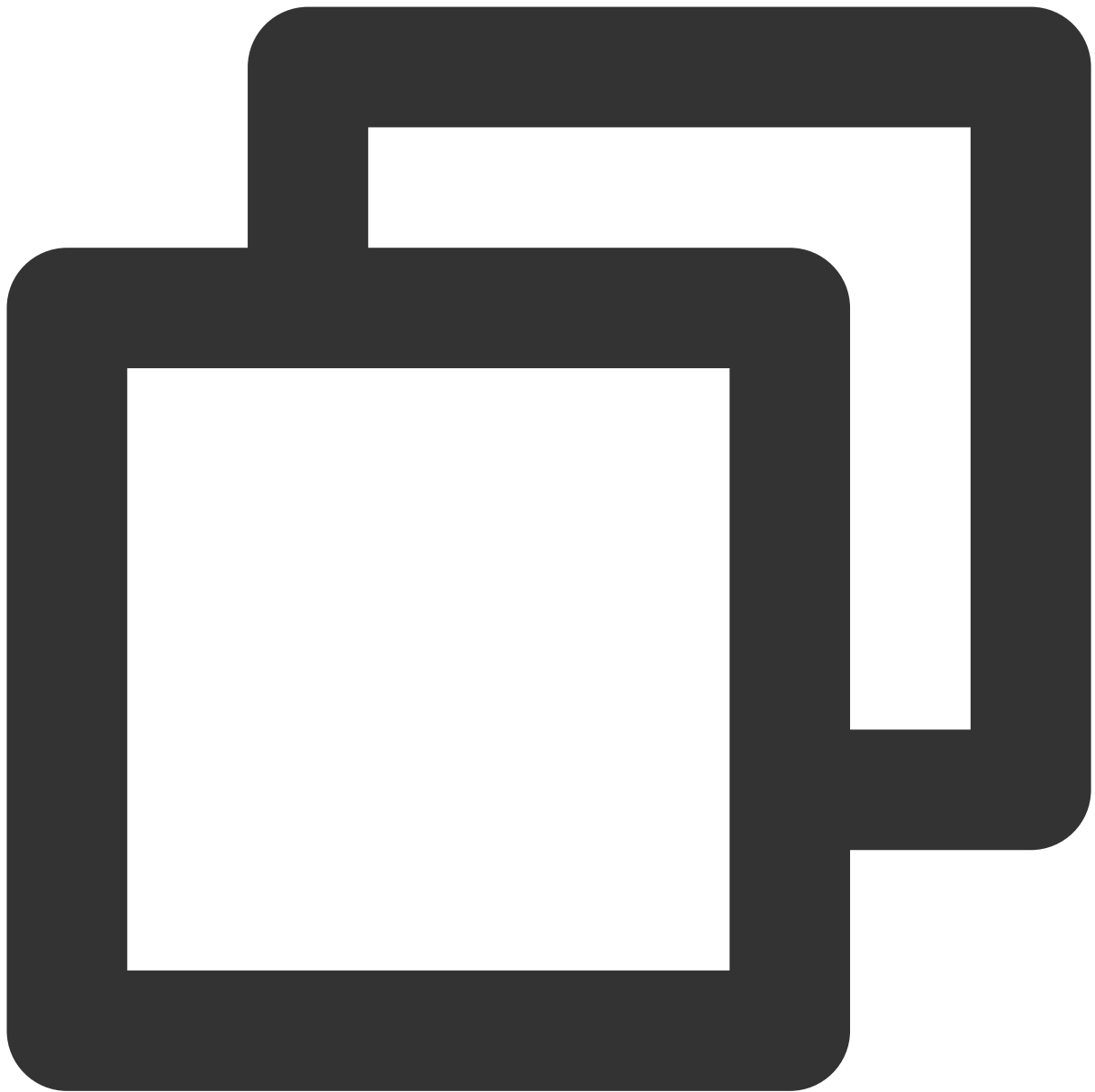
宿主应用开发者可以通过 SDK 提供的方法设置用户的属性信息（地区或者账号），方便在灰度推送等场景下使用。

设置用户 ID

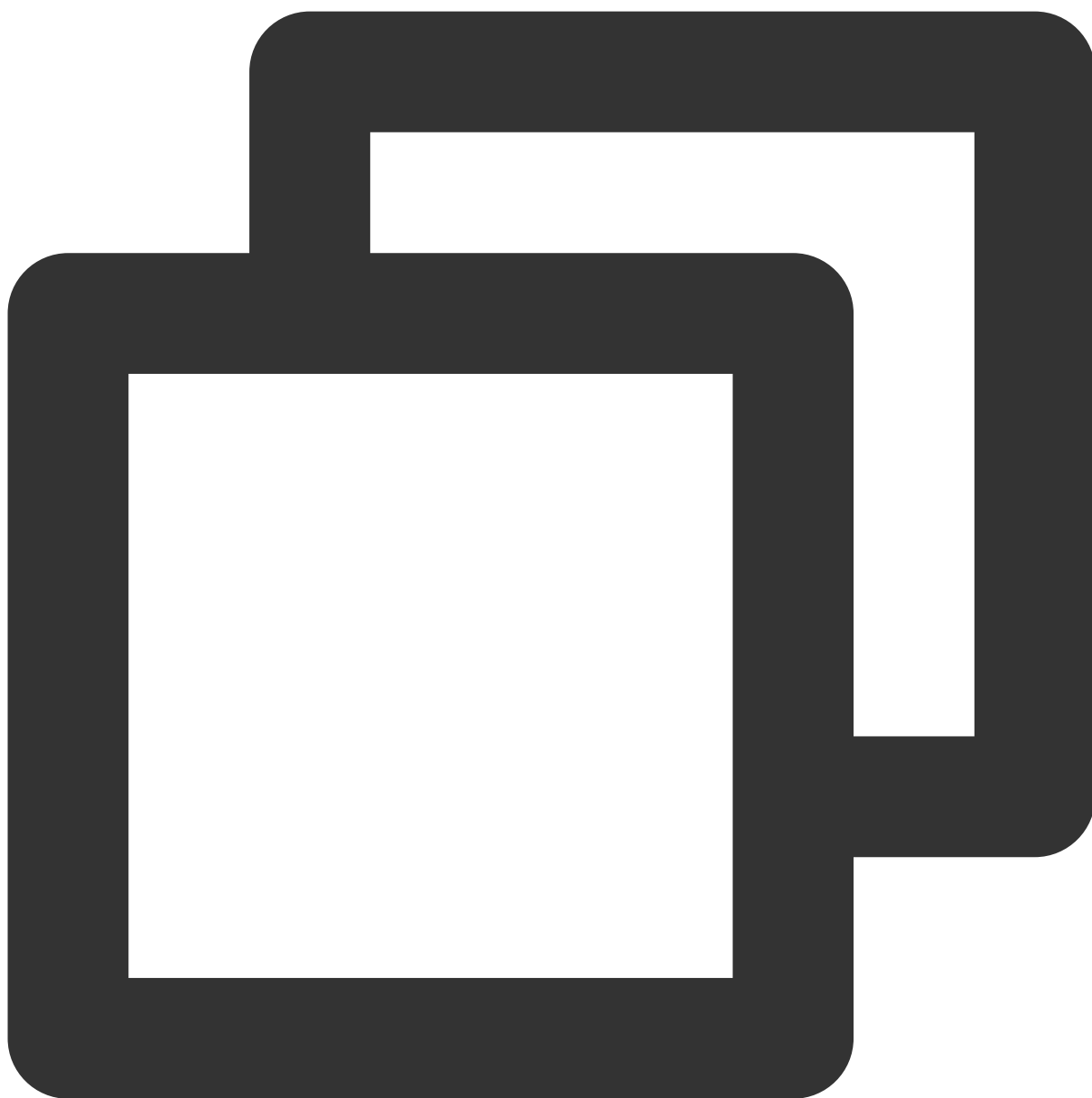


```
// 上报用户ID, 如果参数为空 则视为解绑
// Report user ID, If the parameter is empty, it will be regarded as unbinding
// @param customizedUserID 用户自定义 user ID - customized user ID
- (void)updateCustomizedUserID:(nullable NSString *)customizedUserID;
```

设置位置信息



```
// 上报位置信息
// Report location information
// @param country 国家 - country
// @param province 省 - province
// @param city 市 - city
- (void)updateAreaInfoWithCountry:(nullable NSString *)country Province:(nullable N
```



```
[[TMFMiniAppSDKManager sharedInstance] updateCustomizedUserID:@"abc123"];  
[[TMFMiniAppSDKManager sharedInstance] updateAreaInfoWithCountry:@"中国" Province:@""]
```

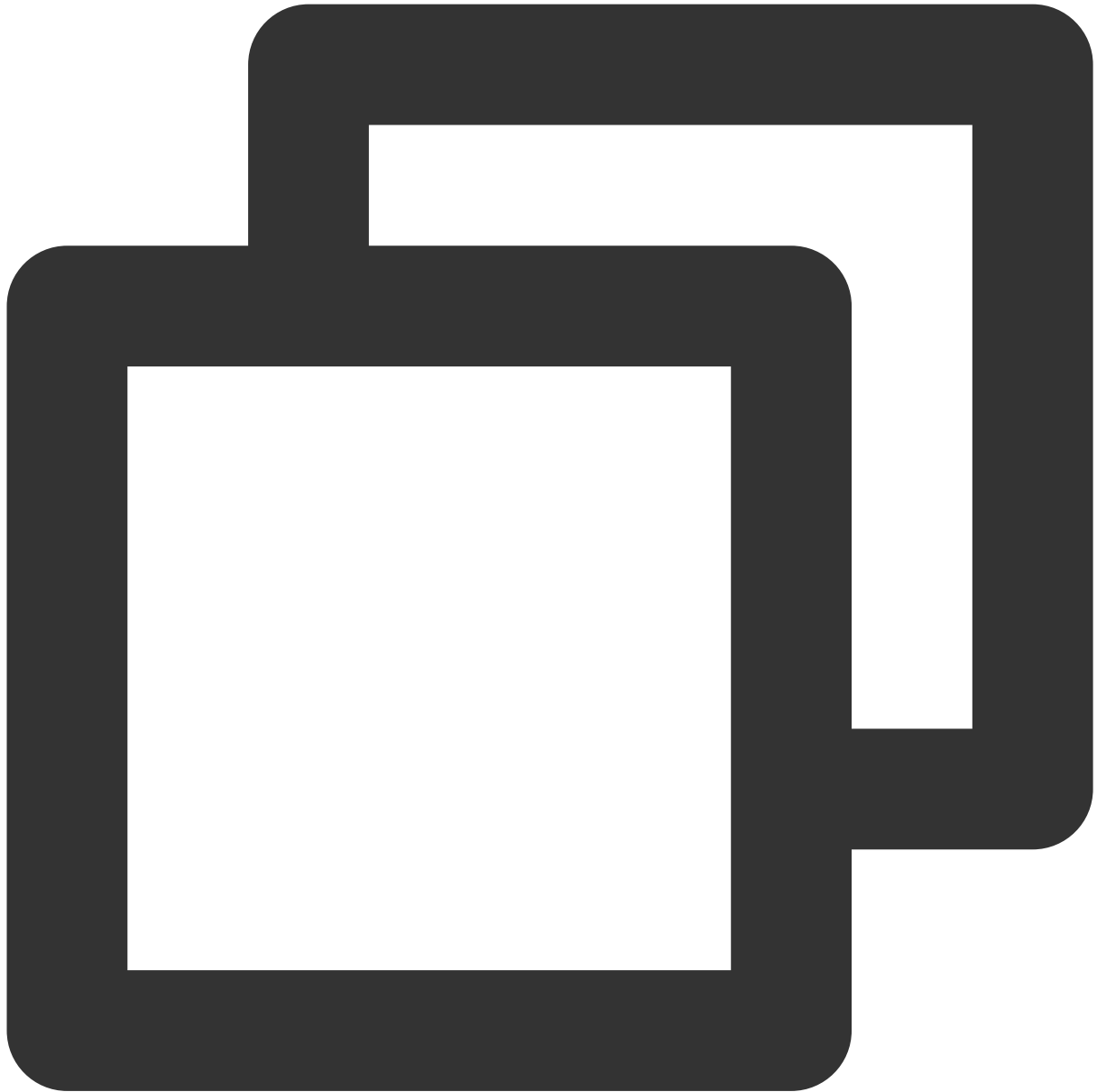
是否支持数据上报时上报到三方服务？

支持 App 将小程序数据上报到任意服务，详细可参见[日志与事件上报](#) 部分。

小程序是否支持不同登录用户的数据隔离？

支持。

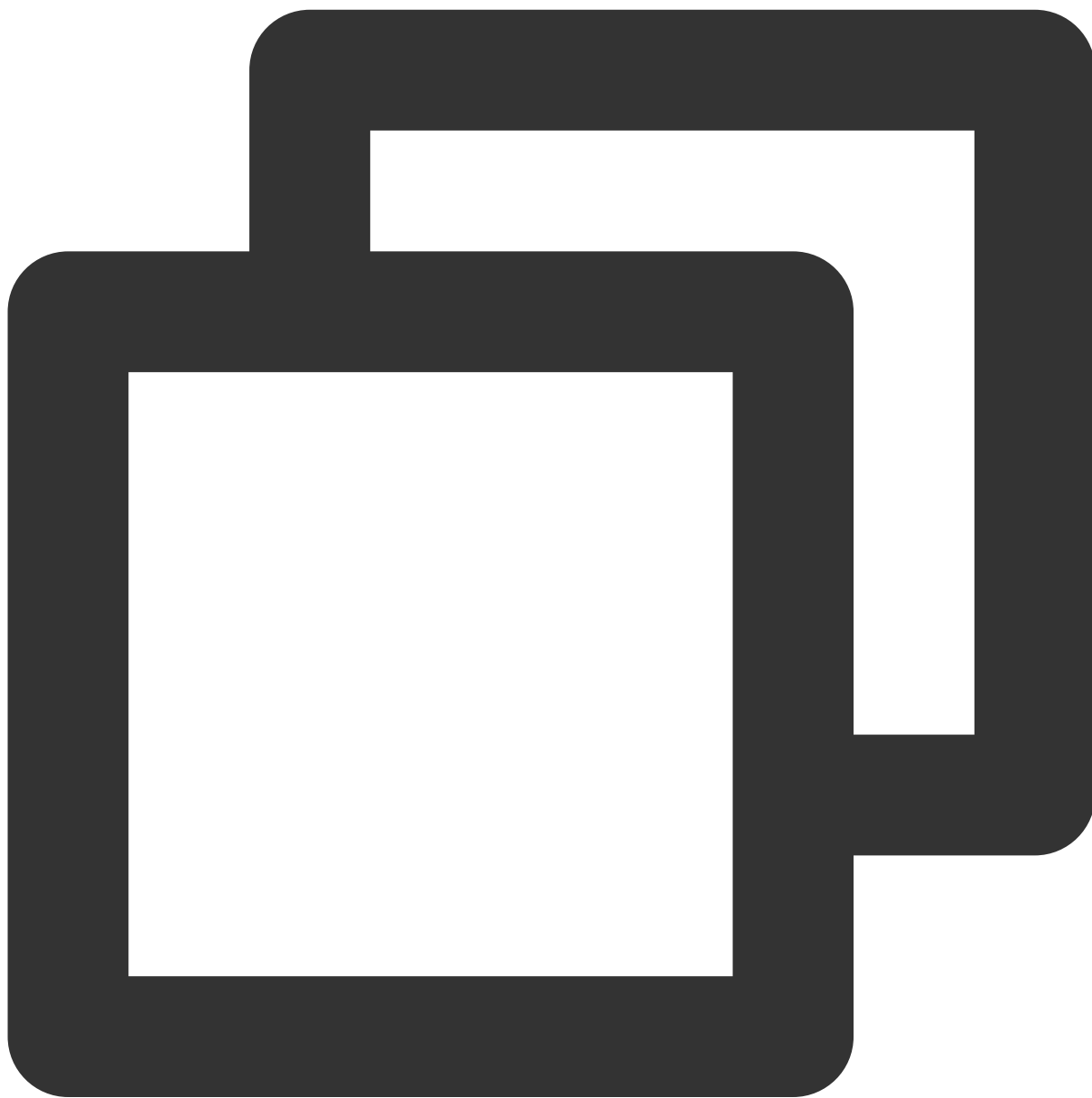
TCMPP SDK 支持小程序的数据根据不同的登录用户进行隔离存储，进行数据保护及业务逻辑的混乱，需要开发者在代理中实现返回AppUID，SDK 根据 AppUID 进行数据存储。



```
/**
 * @brief 获取SDK宿主平台的当前用户账号标识，一般填uin或openid - Get the current user account identifier
 *
 * 注意：返回nil会导致SDK内某些缓存失效。如果没有登录，可以填个设备号id来避免缓存失效 - Note: Return nil will cause some cache in SDK to be invalid. If not logged in, you can fill in a device ID to avoid cache invalidation.
 */
- (NSString *_Nonnull)getAppUID;
```

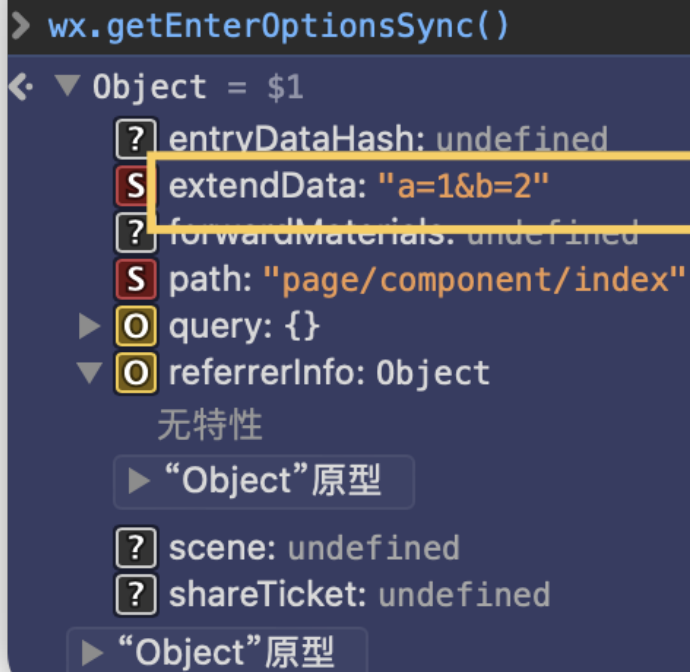
小程序打开时如何传递参数？

打开小程序时可以通过指定paramsStr参数：



```
[[TMFMiniAppSDKManager sharedInstance] startUpMiniAppWithAppID:@"mpbz0uqwzddj5z  
  
}];
```

在小程序中可以在 `wx.getEnterOptionsSync` 的 `extendData` 字段获取打开时传入的参数：



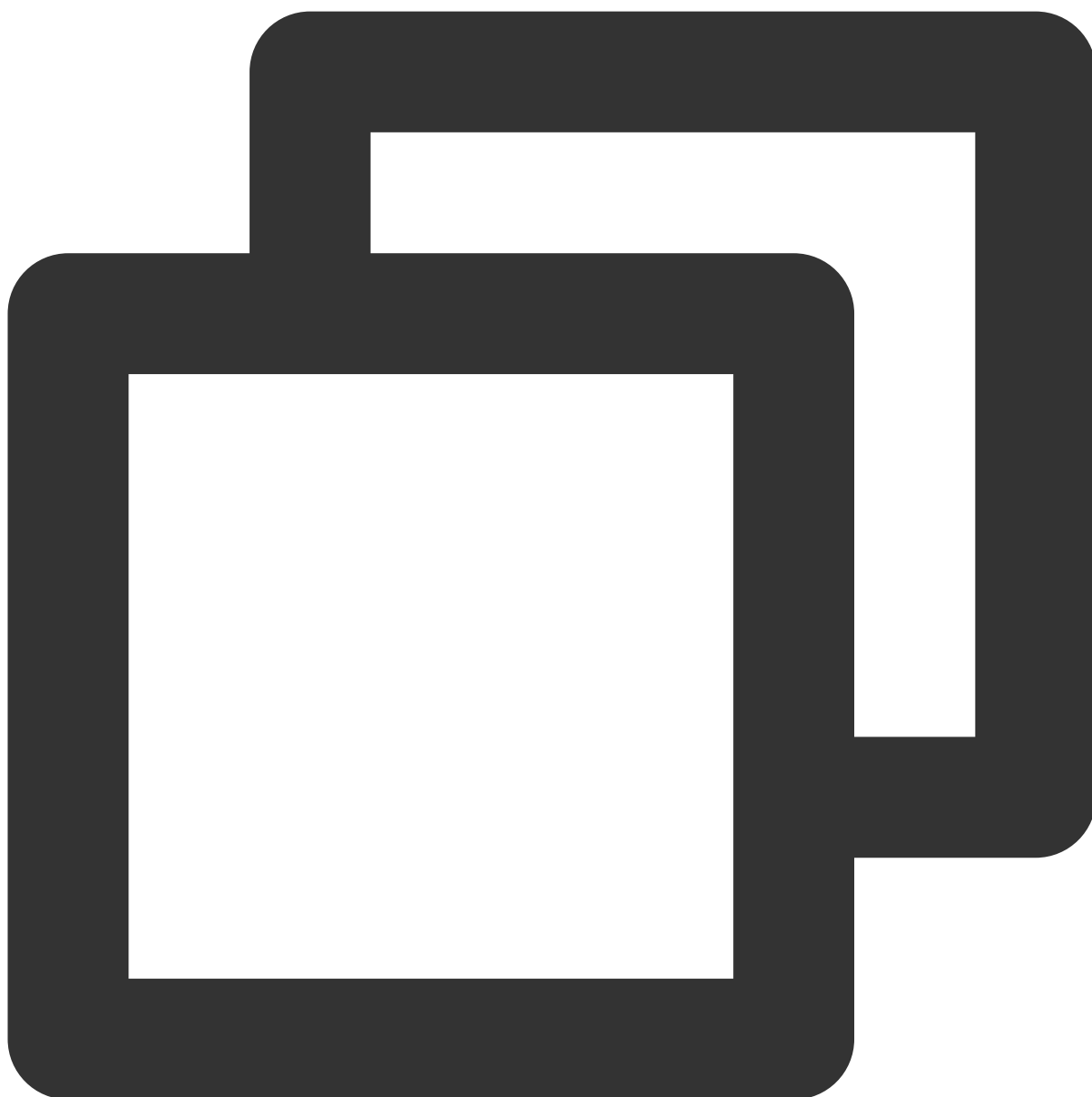
```
> wx.getEnterOptionsSync()
< ▼ Object = $1
  ? entryDataHash: undefined
  S extendData: "a=1&b=2"
  ? forwardMaterials: undefined
  S path: "page/component/index"
  ▶ O query: {}
  ▼ O referrerInfo: Object
    无特性
    ▶ “Object”原型
  ? scene: undefined
  ? shareTicket: undefined
  ▶ “Object”原型
```

调试相关

如何调试小程序？

在集成小程序 SDK 后，Xcode 运行起小程序后（使用 develop profile 打包的应用也可以使用电脑 Safari 查看），可以打开电脑 Safari，然后在工具栏开发 中，选中运行的模拟器或者真机设备，就可以选择小程序打开的页面列表，选中当前打开的页面，就可以审核页面元素、查看网络调用，以及一些 log 日志等。

在 ios16.4 后，因为系统限制，需要打开 inspectable 才能支持进行 safari 开发模式，可以在代理类中实现相关接口，返回 YES。

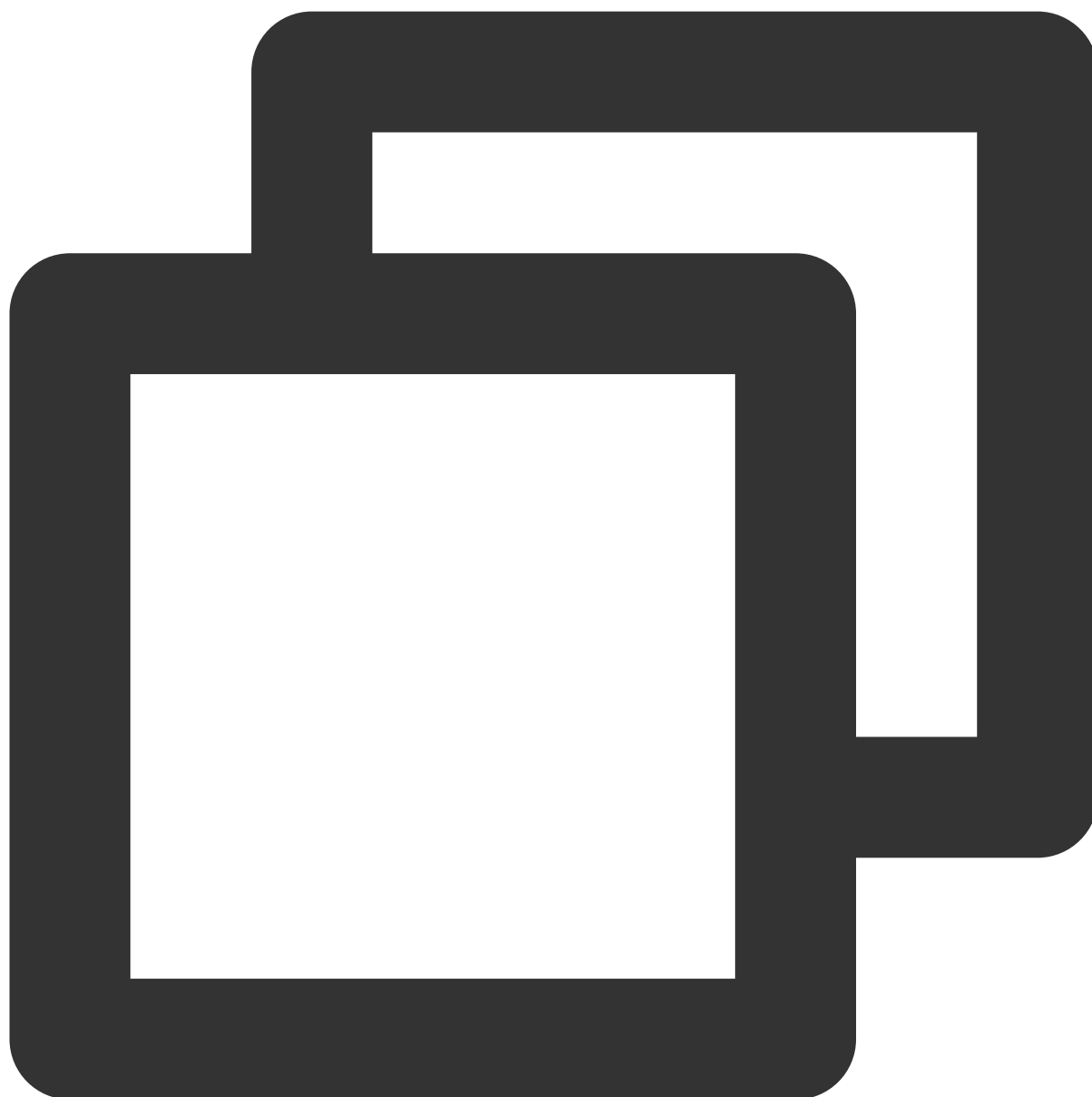


```
// ios16.4后是否开启inspectable, 可以通过 safari 进行开发调试
// Whether to enable inspectable after ios16.4, you can develop and debug through s
- (BOOL)inspectableEnabled;
```

如何开启 vConsole ?

开发版与预览版小程序支持通过胶囊菜单中打开调试开关查看 VConsole 信息，正式版本小程序无法打开 VConsole，需要小程序内部通过调用 `wx.setEnableDebug` 来打开 VConsole。

TCMPP SDK 也支持在 App 设置自动打开所有小程序的 vConsole 模式，需要在代理类中实现相关的接口。



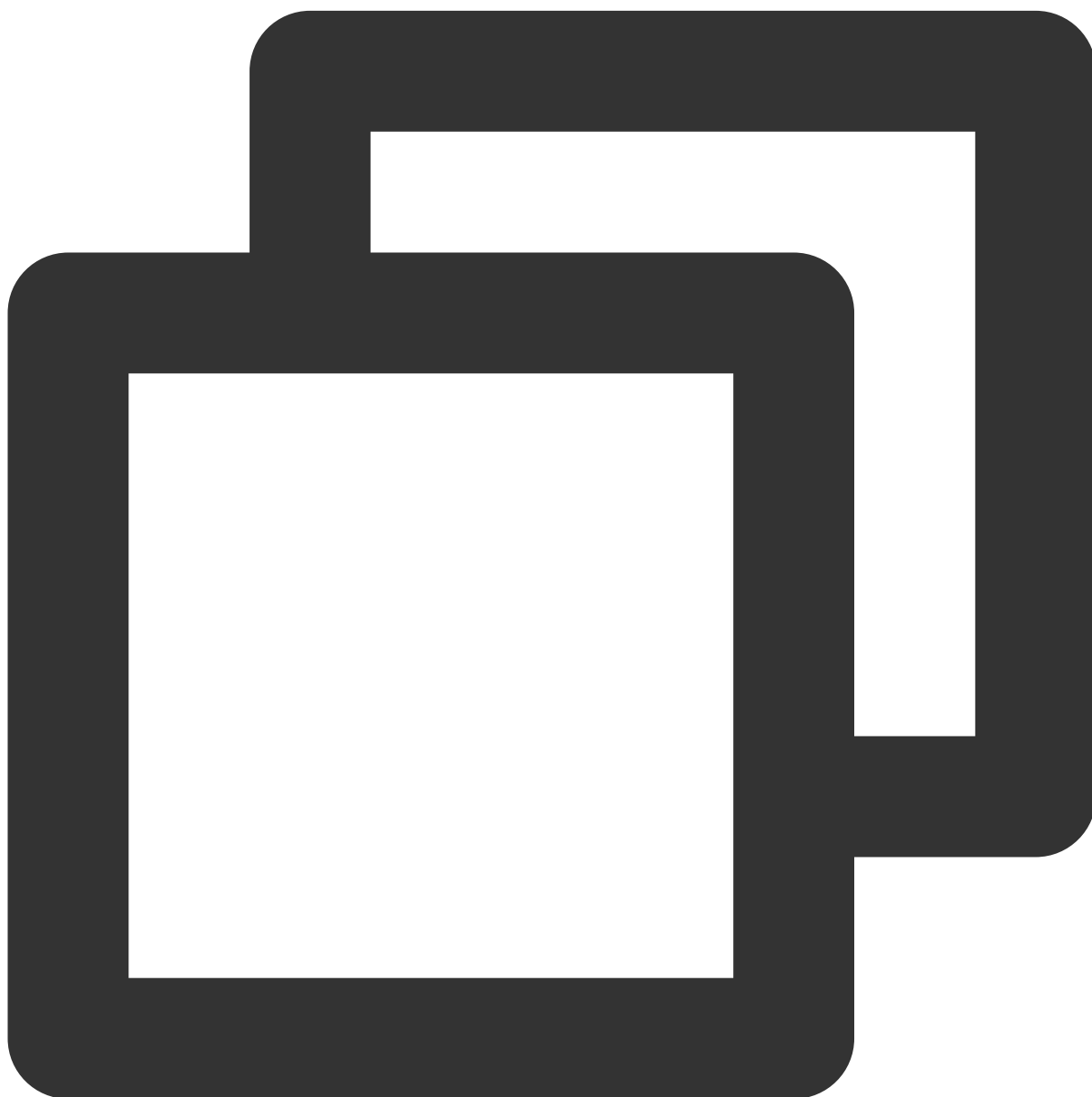
```
// 调试是否开启
// Whether debugging is enabled
- (BOOL)vConsoleEnabled;
```

iOS 错误码

最近更新时间：2024-07-24 10:06:26

打开小程序错误码

返回错误码描述。



// 初始化信息错误或者未初始化

```
// Initialization information is incorrect or not initialized
TMAMiniAppErrorConfigInitError          = -21000,

// 网络请求错误
// The network error
TMAMiniAppErrorNetworkError              = -21001,

// 服务端返回code错误
// The server returns an error code
TMAMiniAppErrorResponseCodeError        = -21002,

// 服务端返回response为空
// The server returns an empty response
TMAMiniAppErrorResponseDataNull         = -21003,

// 小程序版本不存在
// The mini program version does not exist
TMAMiniAppErrorVersionNotExist           = -21005,

// 小程序已下架
// The mini program has been removed from the shelves
TMAMiniAppErrorForbidden                 = -21006,

// 特定接口调用频率过高
// The frequency of specific interface calls is too high
TMAMiniAppCmdIDRequestFrequent           = -21011,

// 接口调用频率过高
// Interface calls are too frequent
TMAMiniAppRequestFrequent                 = -21012,

// 不支持的链接
// Unsupported link
TMAMiniAppErrorErrorLink                 = -21101,

// 未集成TCMPPExtScanCode扩展库
// Unsupported link
TMAMiniAppErrorNoScanCodeSDK             = -21102,

// delegate没有实现
// delegate is not implemented
TMAMiniAppErrorNotImplemented            = -21103,

// 小程序基础库下载失败
// Mini program base library download failed
TMAMiniAppErrorJSSDKDownloadFail         = -22001,
```

```
// 小程序基础库解压失败
// Mini program base library decompression failed
TMAMiniAppErrorJSSDKUnzipFail = -22002,

// 小程序基础库不需要升级
// Mini program base library does not need to be upgraded
TMAMiniAppErrorJSSDKNeedUpgrade = -22003,

// 小程序基础库验证失败
// Mini program basic library verification failed
TMAMiniAppErrorJSSDKVerifyFail = -22004,

// 下载小程序apkg包的网络响应无效
TMAMiniAppErrorResponseInvalid = -22010,

// 解压小程序Apkg包失败
// UnPack mini program Apkg package failed
TMAMiniAppErrorUnpackApkgFail = -22011,

TMAMiniAppErrorConsistencyInvalid = -22012,

TMAMiniAppErrorHTTPStatusInvalid = -22013,

// 小程序Apkg包下载存储路径错误
// The download storage path of the mini program Apkg package is wrong
TMAMiniAppErrorCreatePathFail = -22014,

// 验证小程序Apkg包失败
// The download storage path of the mini program Apkg package is wrong
TMAMiniAppErrorValidateFileFail = -22015,

// 设备系统不支持小程序
// The device system does not support mini programs
TMAMiniAppErrorSystemNotSupport = -22017,

// 启动小程序的AppInfo错误
// AppInfo error when starting the mini program
TMAMiniAppErrorAppInfoError = -22018,

// 小程序已经在显示中
// The mini program is already being displayed
TMAMiniAppErrorMiniAppIsShowing = -22019,

// 小程序视图是无效视图
// The mini program view is an invalid view
TMAMiniAppErrorParentVCInvalid = -22020,
```

```
// 启动小程序的 Application 错误
// Application error when starting the mini program
TMAMiniAppErrorApplicationError = -22021,

// 启动小程序的代理没有实现
// The agent for starting the mini program is not implemented

TMAMiniAppErrorDelegateError = -22022,
TMAMiniAppErrorNilApplication = -22023,
TMAMiniAppErrorUserNotLoggedIn = -22024,
TMAMiniAppErrorFlutterResourceInvalid = -22025,

// 小程序读取资源文件发生异常 (如小游戏读取不到game.json)
// An exception occurs when the mini program reads the resource file (for example)
TMAMiniAppErrorReadPackageFail = -22027,

// 下载文件md5验证失败
// Download file md5 verification failed
TMAMiniAppErrorFileMd5VerifyFail = -22029,

// 复制域名配置文件到目标位置失败
// Failed to copy domain name configuration file to target location
TMAMiniAppErrorFCopyDomainsFileFail = -22030,

// MAWebView 的白屏错误
// MAWebView white screen error
TMAMiniAppErrorWebViewNoCompositingView = -23001,
TMAMiniAppErrorWebViewJSError = -23002,
TMAMiniAppErrorWebViewDidTerminate = -23003,
TMAMiniAppErrorWebViewScriptAgainError = -23004,
TMAMiniAppErrorWebViewPageNotFound = -23005,
TMAMiniAppErrorWebViewApplicationNotFound = -23006,

TMAMiniAppErrorHotlaunchParamInvalid = -23007,
TMAMiniAppErrorWebViewRenderBlank = -23008,

// 无可用的云资源使用
// No available cloud resources to use
TMAMiniAppErrorResourceLimited = -25001,
TMAMiniAppErrorResourceOverflow = -25002
```

iOS 隐私合规

最近更新时间：2024-07-03 18:16:31

小程序SDK涉及的系统权限信息：

SDK名称	涉及的系统权限	权限使用描述
TCMPPSDK	NSPhotoLibraryAddUsageDescription NSCameraUsageDescription NSMicrophoneUsageDescription	用于实现拍照、音频录入、保存图片、保存视频的功能
TCMPPExtMedia	NSPhotoLibraryUsageDescription	用于选择照片和视频
TCMPPExtScanCode	NSCameraUsageDescription	用于扫描二维码
TCMPPExtQMap	NSLocationAlwaysAndWhenInUseUsageDescription NSLocationUsageDescription NSLocationAlwaysUsageDescription NSLocationWhenInUseUsageDescription	用于定位和使用腾讯地图服务
TCMPPExtBaiduMap	NSLocationAlwaysAndWhenInUseUsageDescription NSLocationUsageDescription NSLocationAlwaysUsageDescription NSLocationWhenInUseUsageDescription	用于定位和使用百度地图服务
TCMPPExtAMap	NSLocationAlwaysAndWhenInUseUsageDescription NSLocationUsageDescription NSLocationAlwaysUsageDescription NSLocationWhenInUseUsageDescription	用于定位和使用高德地图服务
TCMPPExtLive	NSCameraUsageDescription NSMicrophoneUsageDescription	用于实现实时音视频录入功能
TCMPPExtAuthentication	NSFaceIDUsageDescription	用时使用设备生物认证功能
TCMPPExtBLE	NSBluetoothAlwaysUsageDescription NSBluetoothPeripheralUsageDescription	用于实现蓝牙相关功能，如蓝牙设备连接等
TCMPPExtCalendar	NSCalendarsUsageDescription NSRemindersUsageDescription	用于实现日期选择和添加提醒事项功能

TCMPPExtClipBoard	无	用于实现访问粘贴板
TCMPPExtContact	NSContactsUsageDescription	用于获取、添加联系人功能
TCMPPExtLBS	NSLocationAlwaysAndWhenInUseUsageDescription NSLocationUsageDescription NSLocationAlwaysUsageDescription NSLocationWhenInUseUsageDescription NSMotionUsageDescription	用于实现定位、系统地图和运动相关功能
TCMPPExtMDNS	NSBonjourServices	用于提供局域网通信的能力
TCMPPExtNetwork	NSAppTransportSecurity NSAllowsArbitraryLoads NSBonjourServices	用于提供 TCP/UDP 通信的能力
TCMPPExtMp3Encoder	无	用于音频MP3格式编码

更多信息，可参见 [隐私合规说明](#)。

Flutter

Flutter SDK 说明

SDK 介绍

最近更新时间：2024-08-06 16:29:51

TCMPP 介绍

TCMPP 由管理控制台，tcmpp_flutter 和小程序开发工具（TCMPP IDE）三个部分组成。

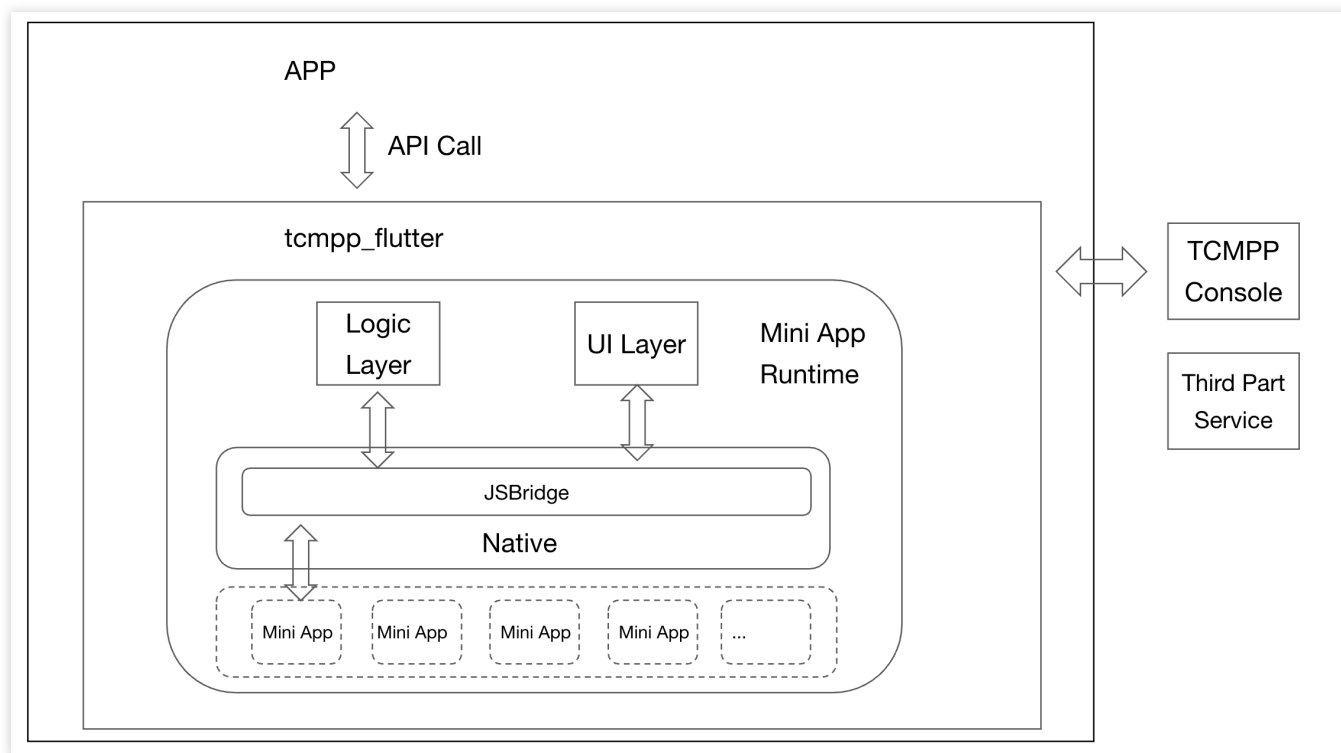


管理控制台：表示TCMPP控制台；包含有小程序管理、客户端应用管理、小程序运维管理等能力。

客户端：表示集成 tcmpp_flutter 的移动客户端应用程序；tcmpp_flutter 为客户端应用提供小程序的运行环境。

开发者工具：小程序开发者使用小程序 IDE 进行小程序的开发、调试和版本提交；小程序开发者使用小程序预览助手，在移动客户端进行小程序预览验证。

tcmpp_flutter 工作原理



`tcmpp_flutter` 给客户端应用提供小程序的运行时环境；`tcmpp_flutter` 通过和 TCMPP 后台通信，获取小程序信息，并在 `tcmpp_flutter` 提供的运行时环境中加载和运行小程序。

`tcmpp_flutter` 出于减少包大小和减少系统权限及功能控制的考虑，把插件分为了核心库（即`tcmpp_flutter`）和扩展库（即`tcmpp_flutter_xxx`），大部分功能只需要集成核心库就可以实现，并且不需要多余的系统权限申请。只集成核心库对最终 App 安装包的影响在4M左右。

扩展库的相关说明请参考[扩展插件](#)文档。

SDK 快速集成

最近更新时间：2024-08-06 16:29:21

集成样例代码

您可以 [通过插件官网获取Demo地址](#)。

前置条件

环境要求

sdk: '>=3.0.0 <4.0.0'

flutter: '>=3.3.0'

插件依赖

plugin_platform_interface

flutter_plugin_android_lifecycle

集成方式

从 Pub.dev 添加 TCMPP 核心插件

1.添加依赖关系要从[pub.dev](#) 添加插件，请打开应用程序文件夹内的 pubspec.yaml 文件，并在依赖关系下添加 `tcmpp_flutter: ${version}`。

```
22  sdk: ^3.2.0 <4.0.0
23
24  # Dependencies specify other packages that your package needs in order to
25  # To automatically upgrade your package dependencies to the latest version
26  # consider running `flutter pub upgrade --major-versions`. Alternatively,
27  # dependencies can be manually updated by changing the version numbers to
28  # the latest version available on pub.dev. To see which dependencies have
29  # versions available, run `flutter pub outdated`.
30  dependencies:
31    flutter:
32      sdk: flutter
33
34    tcmpp_flutter: ^1.0.0
35
36
```

2.安装依赖关系

从终端 运行 `flutter pub get`。

从 VS 代码：单击位于 `pubspec.yaml` 顶部操作功能区右侧、由下载图标指示的 Get Packages（获取软件包）。

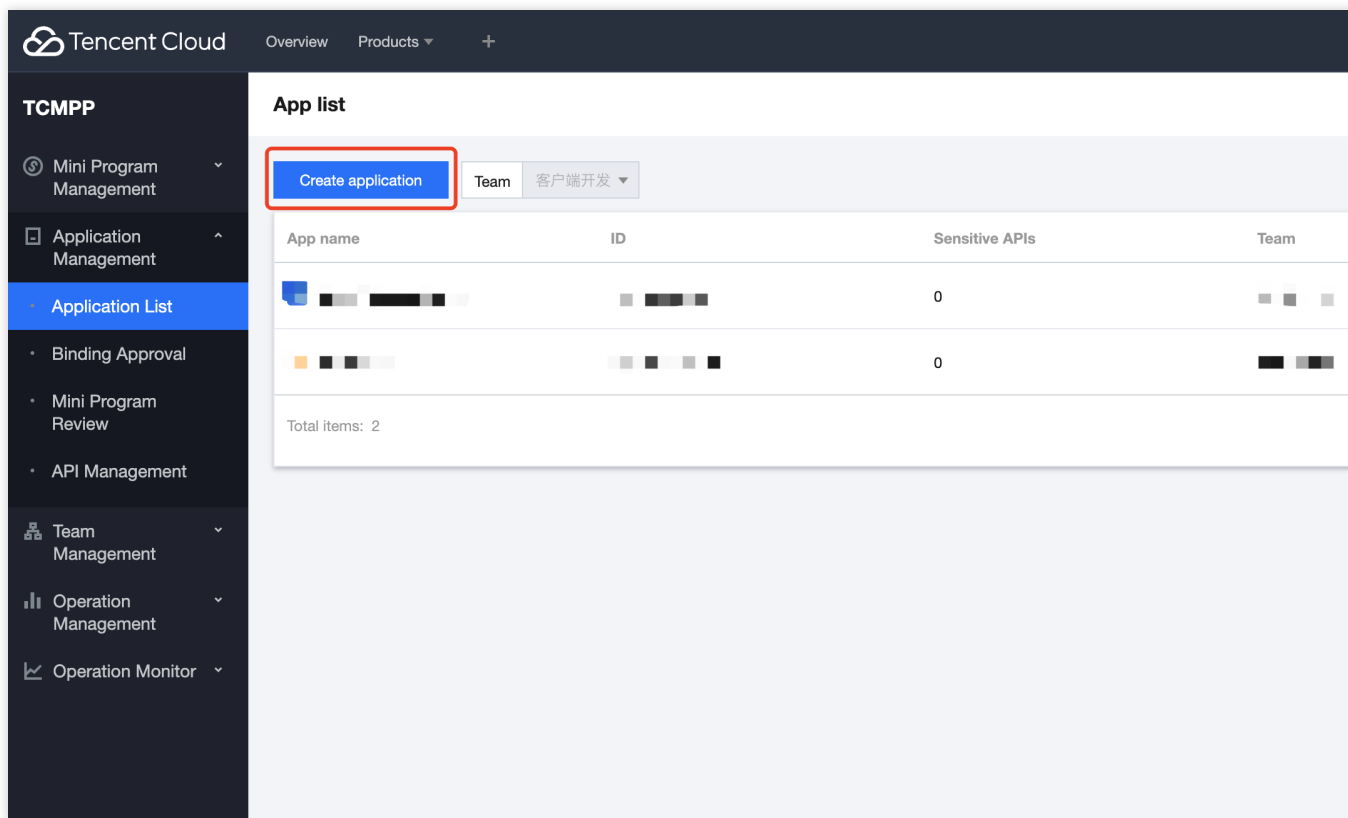
从 Android Studio/IntelliJ：单击 `pubspec.yaml` 顶部操作功能区中的 Pub get。

有关如何添加 Flutter 插件或软件包的详细信息，请参阅[Use packages & plugins](#)。

获取配置文件

小程序 SDK 的初始化依赖于从小程序控制台获取的小程序 SDK 配置文件；在开始集成小程序 SDK 前，需要从小程序控制台获取小程序 SDK 配置文件。

登录控制台后，请单击创建 App：



在弹出的创建应用程序窗口中，填写应用程序的相关信息，在集成平台部分，同时勾选 IOS 平台和 Android 平台。然后在软件包名称字段中填写 application ID 或 bundle ID：

1 Create application

>

2 Integrate mini program container

Team *

Example Team

App name *

SuperApp

App overview

Please enter the app description.

App icon

icon_256x256.png



Select image

Delete

Please upload a square image in .jpg and .png format with a resolution of 128*128. The image is less than 2 MB.

If the icon is not uploaded, the system default icon will be used.

Integration platform *

☒ iOS platform

Bundle ID *

com.tencent.tcmp.demo

App download address

Please enter the app download a

☒ Android Platform

Package Name *

com.tencent.tcmp.demo

App download address

Please enter the app download a

Next Step, Integrate the Mini Program Container

单击下一步，在新页面下载 Android / IOS 配置文件：

1

Specify the application package name and obtain the configuration file.

Package Name

com.tencent.tcmpp.demo

Configuration File ⓘ

Download Configuration File

添加配置文件

- 1. 将 TCMPP 配置文件放在 Flutter 项目中的某个位置。
 - 2. 在项目根目录下添加名为 “tcmpp-plugin-settings.json” 的文件。
 - 3. 编辑 pubspec.yaml 文件并为上述文件添加 flutter 资源。
- 在 **tcmpp-plugin-settings.json** 文件中，您可以选择配置 SDK 的设置。

Android：

字段	类型	说明
configAssetsName	string	TCMPP 和 Android 配置文件路径
debug	bool	启用 SDK 调试和日志
enableX5Core	bool	是否使用 TBS 服务（X5 内核）作为微型程序运行时
x5LicenseKey	string	如果启用运行时，则 TBS 服务许可证
X5DocLicenseKey	string	如果启用运行时，TBS 文档服务许可证
x5Core32Url	string	32 位 TBS 运行时的下载地址
x5Core64Url	string	64 位 TBS 运行时的下载地址

iOS:

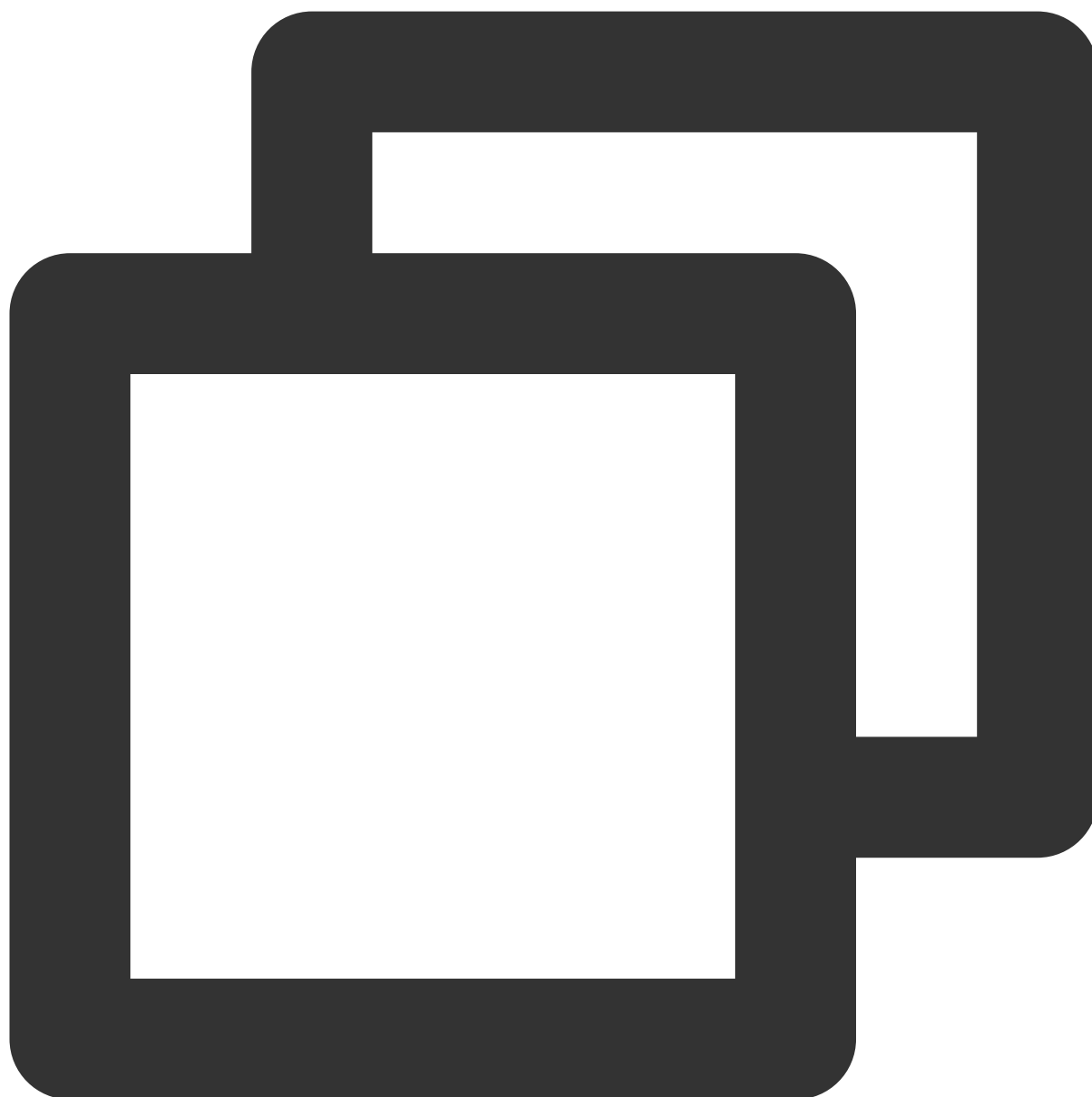
字段	类型	说明
configAssetsName	string	TCMPP iOS 配置文件路径
debug	bool	启用 SDK 调试
logEnable	bool	启用 SDK 日志

inspectableEnabled	bool	iOS 16.4 后启用可检查功能，可通过 safari 调试小程序
--------------------	------	------------------------------------

通用：

字段	类型	说明
appName	string	主机应用程序名称，主要用于小程序中的版权提示
appVersion	string	主机应用程序版本，主要用于小程序中的版权提示

示例内容：

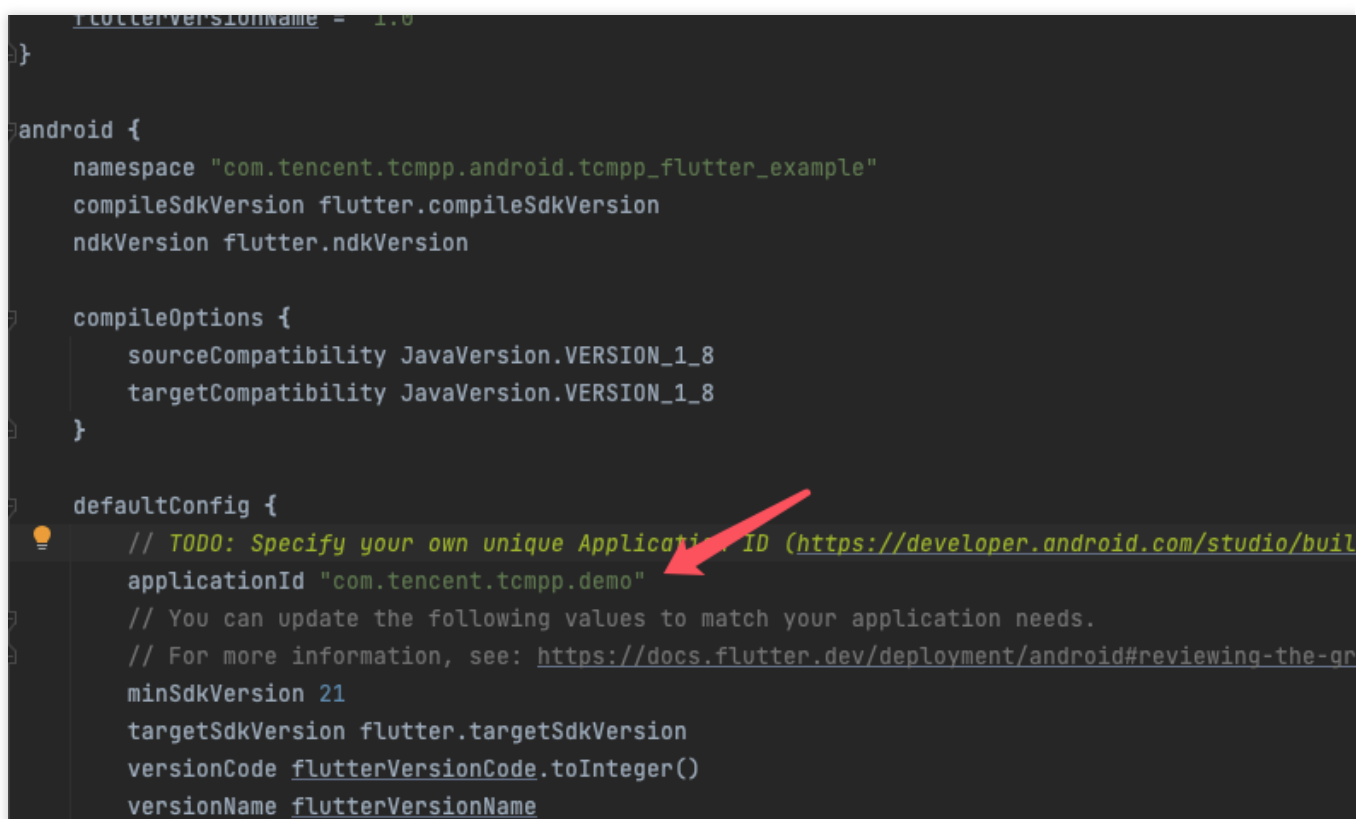


```
{
  "android": {
    "configAssetsName": "tcmpp-android-configurations.json",
    "debug": true,
    "enableX5Core": true,
    "x5LicenseKey": "",
    "x5DocLicenseKey": "",
    "x5Core32Url": "",
    "x5Core64Url": ""
  },
  "ios": {
```



```
"configAssetsName": "tcmpp-ios-configurations.json",
"debug": true,
"logEnable": true,
"inspectableEnabled": true
},
"common": {
  "appName": "tcmpp-flutter",
  "appVersion": "1.0.0"
}
```

对于 Android，在 Flutter 项目中打开 android > app > build.gradle，applicationId 可在 android > defaultConfig 部分找到。



```
FlutterVersionName = 1.0
}

android {
    namespace "com.tencent.tcmpp.android.tcmpp_flutter_example"
    compileSdkVersion flutter.compileSdkVersion
    ndkVersion flutter.ndkVersion

    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }

    defaultConfig {
        // TODO: Specify your own unique Application ID (https://developer.android.com/studio/build)
        applicationId "com.tencent.tcmpp.demo"
        // You can update the following values to match your application needs.
        // For more information, see: https://docs.flutter.dev/deployment/android#reviewing-the-gradle-config
        minSdkVersion 21
        targetSdkVersion flutter.targetSdkVersion
        versionCode flutterVersionCode.toInteger()
        versionName flutterVersionName
    }
}
```

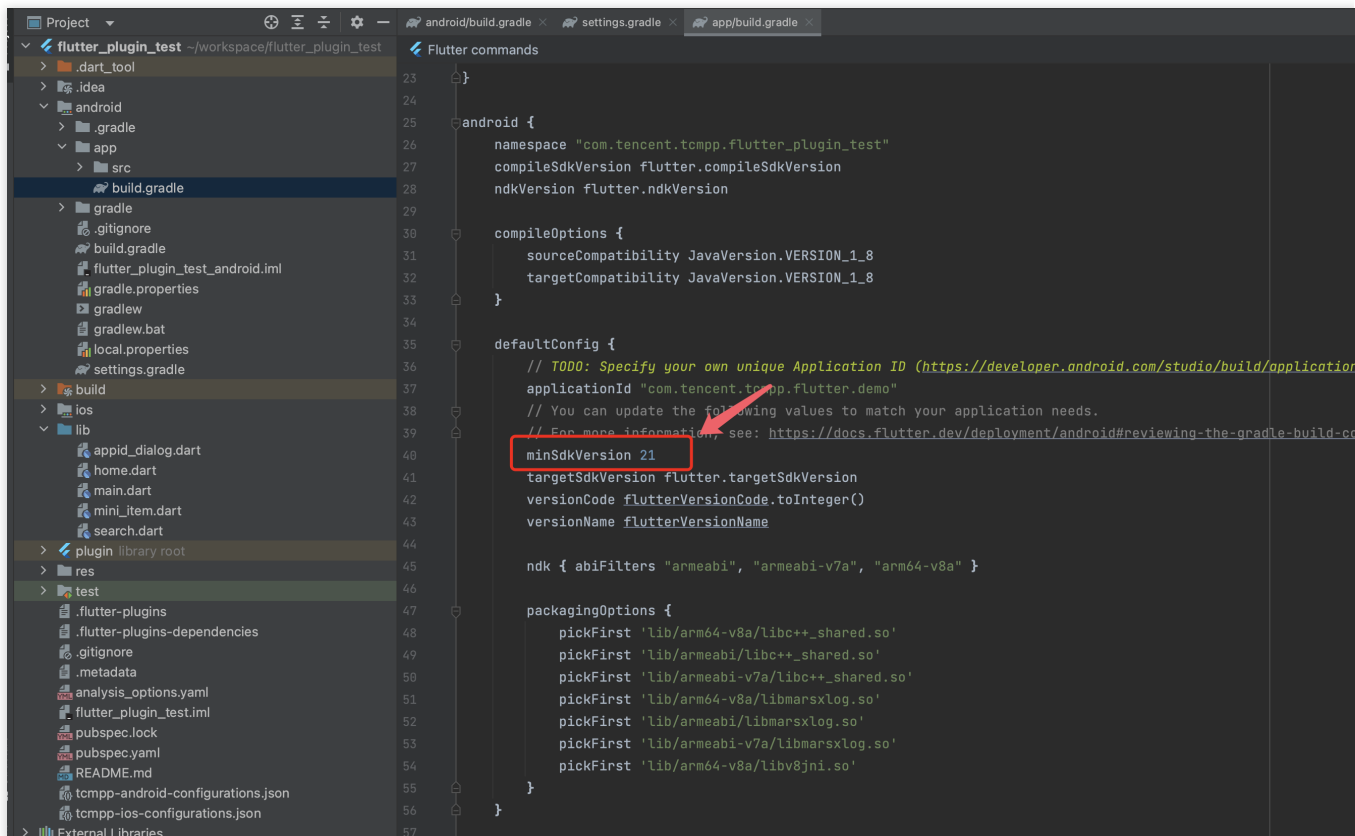
对于 iOS，在 Flutter 项目中打开 ios > Runner.xcodeproj > project.pbxproj，搜索 PRODUCT_BUNDLE_IDENTIFIER。

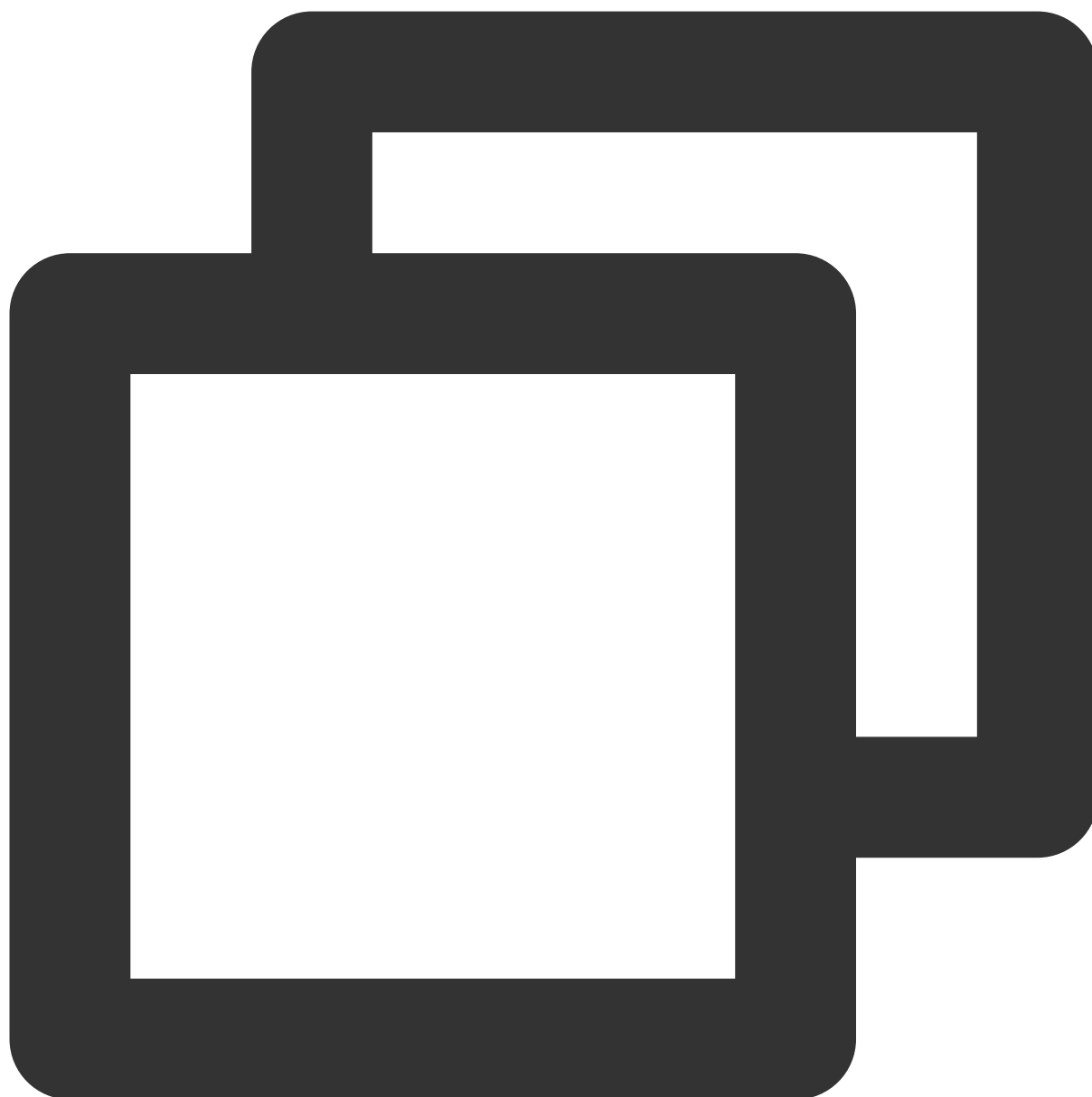
```
example > ios > Runner.xcodeproj > project.pbxproj
424  /* Begin XCBuildConfiguration section */
425  97C147061CF9000F007C117D /* Debug */ = {
426      isa = XCBuildConfiguration;
427      baseConfigurationReference = 9740EEB21CF90195004384FC /* Debug.xcconfig */;
428      buildSettings = {
429          ASSETCATALOG_COMPILER_APPICON_NAME = AppIcon;
430          CODE_SIGN_STYLE = Manual;
431          CURRENT_PROJECT_VERSION = "${FLUTTER_BUILD_NUMBER}";
432          DEVELOPMENT_TEAM = "";
433          "DEVELOPMENT_TEAM[sdk=iphoneos*]" = VU8Y4N38VF;
434          ENABLE_BITCODE = NO;
435          "EXCLUDED_ARCHS[sdk=iphonesimulator*]" = "i386 arm64";
436          INFOPLIST_FILE = Runner/Info.plist;
437          LD_RUNPATH_SEARCH_PATHS = (
438              "$(inherited)",
439              "@executable_path/Frameworks",
440          );
441          PRODUCT_BUNDLE_IDENTIFIER = com.tencent.tcmpp.demo;
442          PRODUCT_NAME = "${TARGET_NAME}";
443          PROVISIONING_PROFILE_SPECIFIER = "";
444          "PROVISIONING_PROFILE_SPECIFIER[sdk=iphoneos*]" = tcmpp_demo;
445          VERSIONING_SYSTEM = "apple-generic";
446      };
447      name = Debug;
448  };
449  97C147071CF9000F007C117D /* Release */ = {
450      isa = XCBuildConfiguration;
451      baseConfigurationReference = 7AFA3C8E1D35360C0083082E /* Release.xcconfig */;
452      buildSettings = {
453          ASSETCATALOG_COMPILER_APPICON_NAME = AppIcon;
454          CODE_SIGN_STYLE = Manual;
455          CURRENT_PROJECT_VERSION = "${FLUTTER_BUILD_NUMBER}";
456          DEVELOPMENT_TEAM = "";
457          "DEVELOPMENT_TEAM[sdk=iphoneos*]" = VU8Y4N38VF;
458          ENABLE_BITCODE = NO;
459          "EXCLUDED_ARCHS[sdk=iphonesimulator*]" = "i386 arm64";
460          INFOPLIST_FILE = Runner/Info.plist;
461          LD_RUNPATH_SEARCH_PATHS = (
462              "$(inherited)",
463              "@executable_path/Frameworks",
464          );
465          PRODUCT_BUNDLE_IDENTIFIER = com.tencent.tcmpp.demo;
466          PRODUCT_NAME = "${TARGET_NAME}";
467          PROVISIONING_PROFILE_SPECIFIER = "";
468          "PROVISIONING_PROFILE_SPECIFIER[sdk=iphoneos*]" = tcmpp_demo;
469          VERSIONING_SYSTEM = "apple-generic";
470      };
471      name = Release;
472  };
473  */
```

特定平台

针对安卓平台

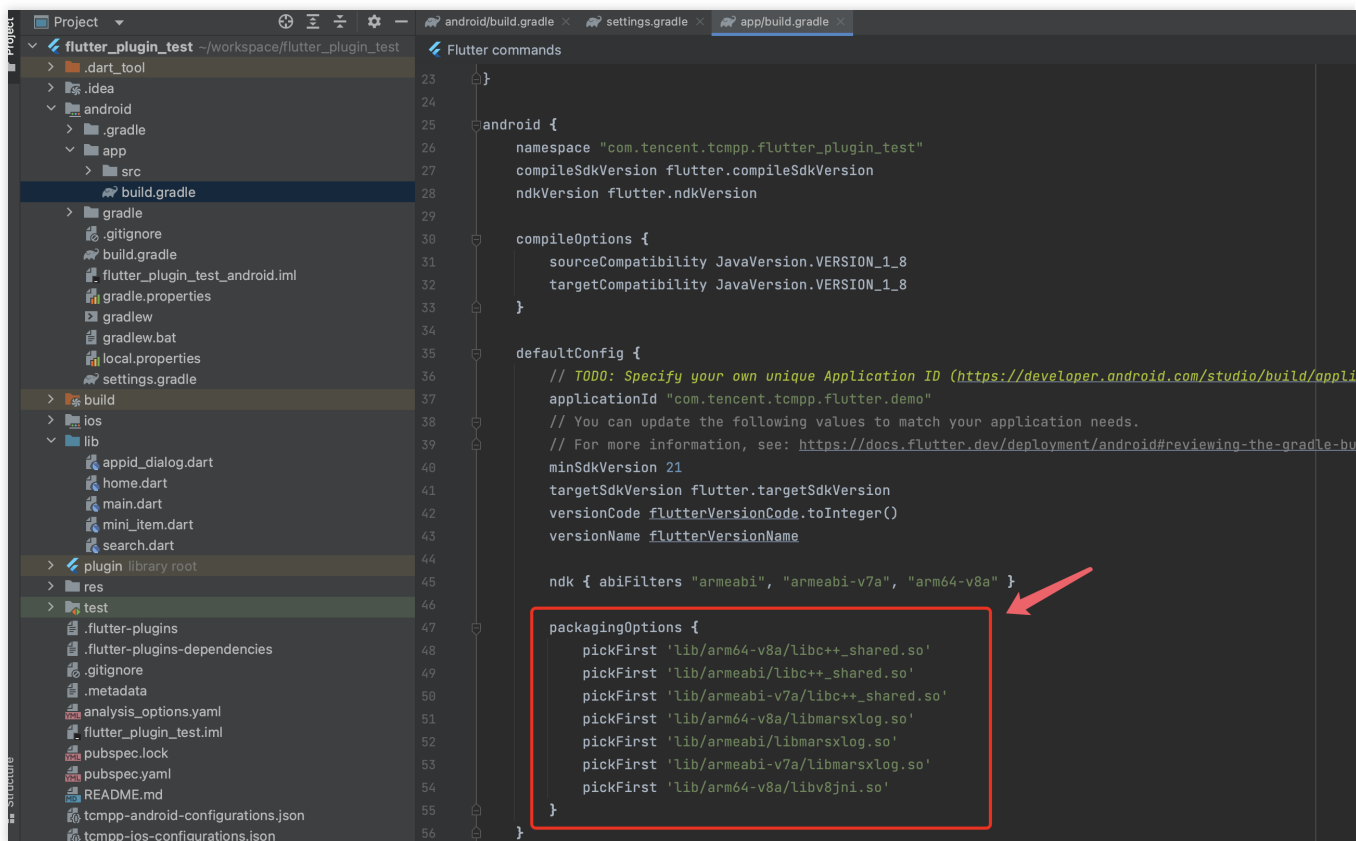
1. 打开 android > app 目录，编辑 build.gradle 文件。
2. 在 android > defaultConfig 部分，将 minSdkVersion 改为不小于 21。

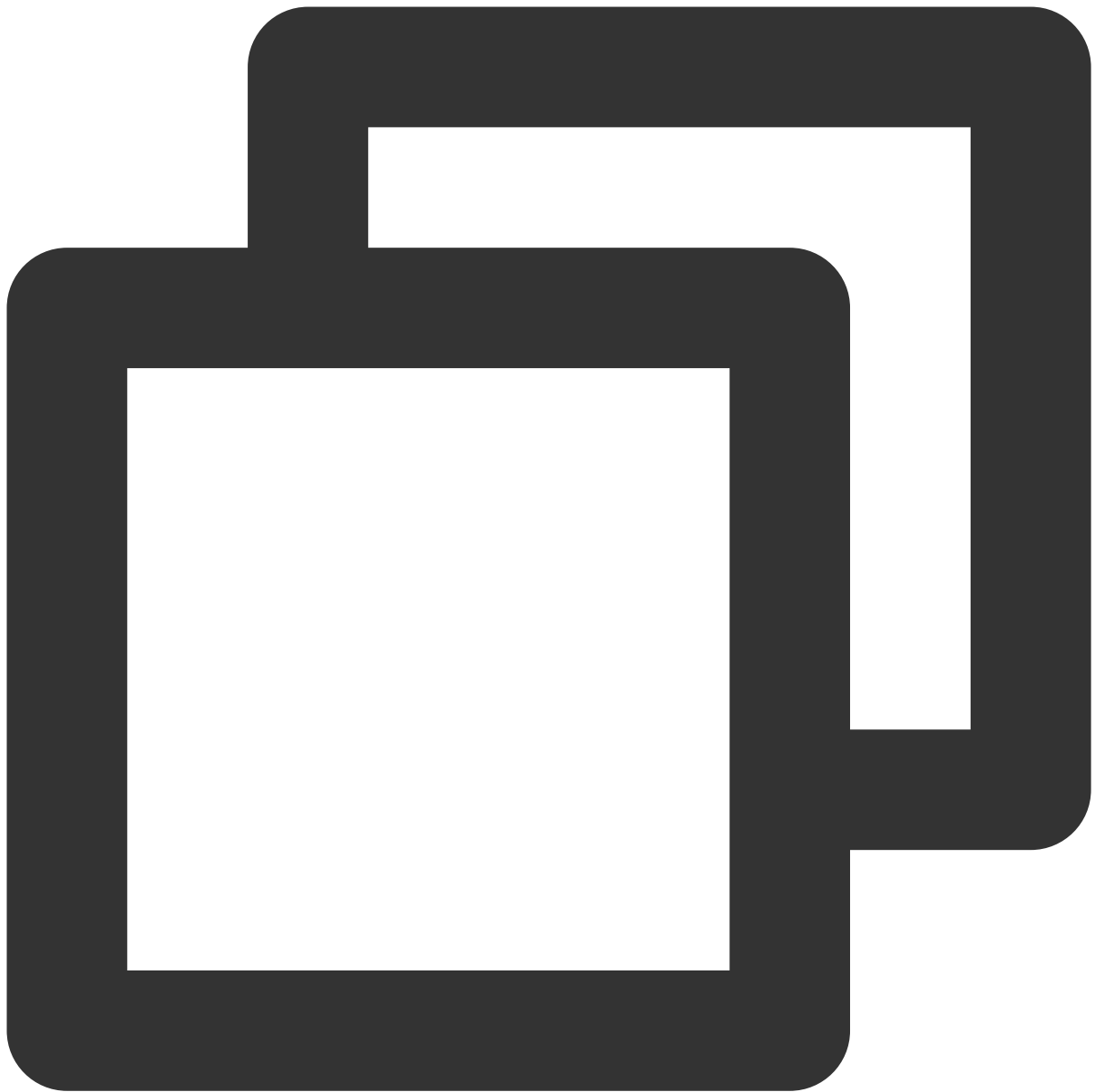




```
minSdkVersion 21
```

3. 在 `android > defaultConfig` 部分，添加 `packagingOptions`。



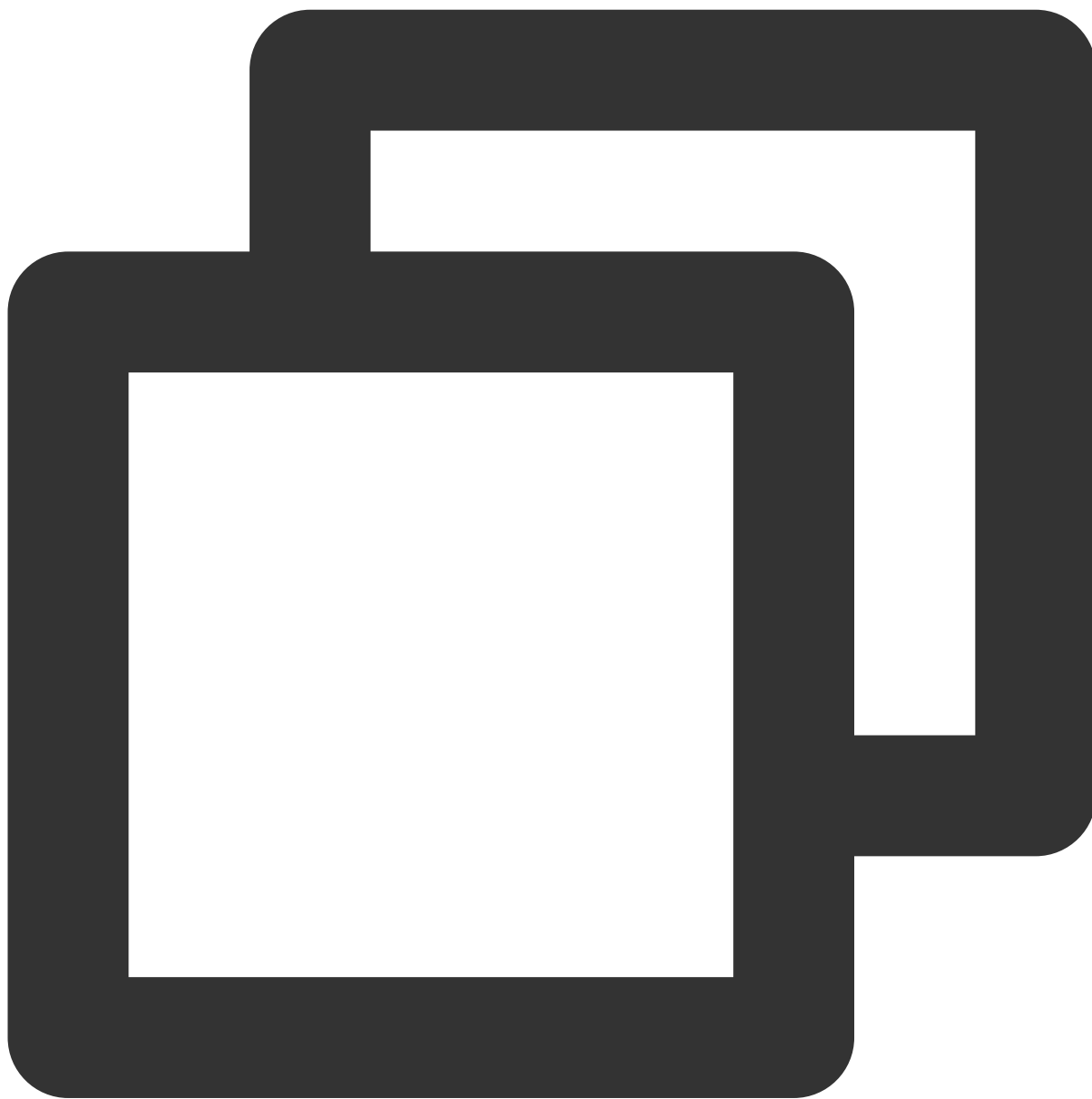


```
packagingOptions {  
  pickFirst 'lib/arm64-v8a/libc++_shared.so'  
  pickFirst 'lib/armeabi/libc++_shared.so'  
  pickFirst 'lib/armeabi-v7a/libc++_shared.so'  
  pickFirst 'lib/arm64-v8a/libmarsxlog.so'  
  pickFirst 'lib/armeabi/libmarsxlog.so'  
  pickFirst 'lib/armeabi-v7a/libmarsxlog.so'  
  pickFirst 'lib/arm64-v8a/libv8jni.so'  
}
```

用于 iOS 平台

添加来源

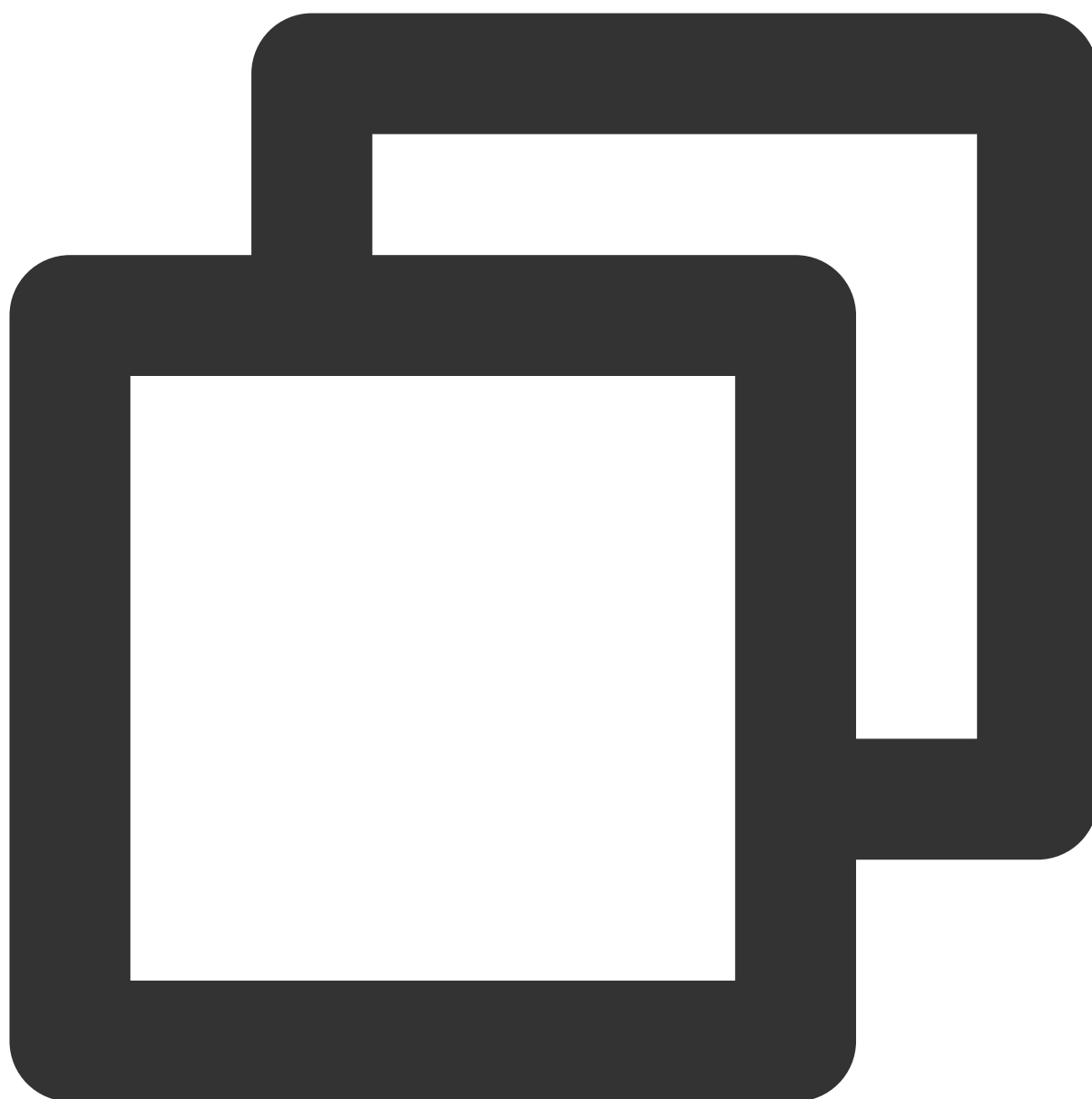
在 ios 目录中的 Podfile 文件中，添加源代码：



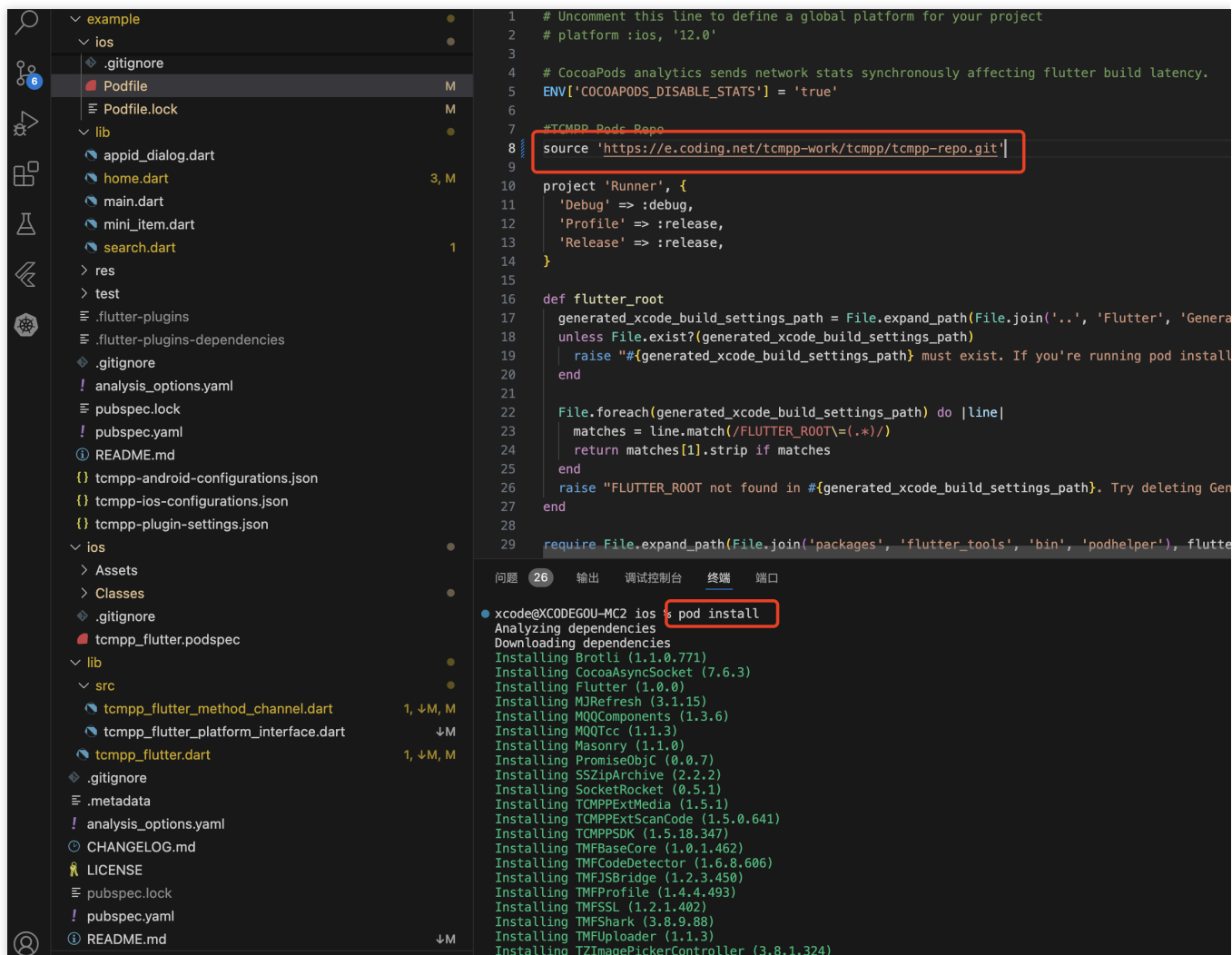
```
source 'https://e.coding.net/tcmpp-work/tcmpp/tcmpp-repo.git'
```

执行 pod 安装

cd 到 ios 目录，执行 pod install：



```
pod install
```

打开小程序

使用 TCMPP API, import 'package:tcmp_flutter/tcmp_flutter.dart'

```
example > lib > main.dart > ...
1  import 'package:flutter/material.dart';
2  import 'package:flutter/services.dart';
3  import 'package:tcmpp_flutter/tcmpp_flutter.dart';
4  import 'package:tcmpp_flutter_example/appid_dialog.dart';
5  import 'package:tcmpp_flutter_example/mini_item.dart';
6  import 'package:tcmpp_flutter_example/search.dart';
7
Run | Debug | Profile
8  void main() {
9    runApp(const MyApp());
10 }
11
12 class MyApp extends StatefulWidget {
13   const MyApp({super.key});
14
15   @override
16   State<MyApp> createState() => _MyAppState();
17 }
18
19 class _MyAppState extends State<MyApp> {
20   @override
21   Widget build(BuildContext context) {
22     SystemChrome.setPreferredOrientations([
23       DeviceOrientation.portraitUp,
24     ]);
25     return const MaterialApp(home: HomePage());
26   }
27 }
28
29 class HomePage extends StatefulWidget {
30   const HomePage({super.key});
31
32   @override
33   State<StatefulWidget> createState() => _HomePageState();
34 }
35
```

然后创建一个 TcmppFlutter 对象，并用它来调用 TCMPP API 打开小程序。

```
class _HomePageState extends State<HomePage> with WidgetsBindingObserver {
  final _tcmppFlutterPlugin = TcmppFlutter();
  final List<AppInfo> _appInfoList = [];

  @override
  void initState() {
    super.initState();
    WidgetsBinding.instance.addObserver(this);
    if (WidgetsBinding.instance.lifecycleState == AppLifecycleState.resumed) {
      getRecentMini();
    }
    _tcmppFlutterPlugin.registerOpenApiHandler(MyOpenApiHandler());
    _tcmppFlutterPlugin.registerPlatformEventHandler(MyPlatformHandler());
    _tcmppFlutterPlugin.registerMiniAppApi("testState", myApiHandler);
    _tcmppFlutterPlugin.registerMiniAppApi("myRequestPayment", myApiHandler1);
    _tcmppFlutterPlugin.setAccount(AccountInfo(
      uid: "12345",
      avatarUrl: "https://picsum.photos/250?image=9",
      accountName: "SimpleAccount"));
    _tcmppFlutterPlugin.setLocale("en", region: "us");
    _tcmppFlutterPlugin.setTheme(MiniTheme.dark);
  }

  @override
  void didChangeAppLifecycleState(AppLifecycleState state) {
    if (state == AppLifecycleState.resumed) {
      getRecentMini();
    }
  }
}
```

```
Future<void> startMiniById(String? appId) async {
  if (appId?.isEmpty ?? false) {
    await _tcmppFlutterPlugin
      .startMiniAppWithId(appId!, null)
      .catchError((err) {
        ScaffoldMessenger.of(context)
          .showSnackBar(SnackBar(content: Text(err.toString())));
      });
  }
}
```

SDK 集成常见问题

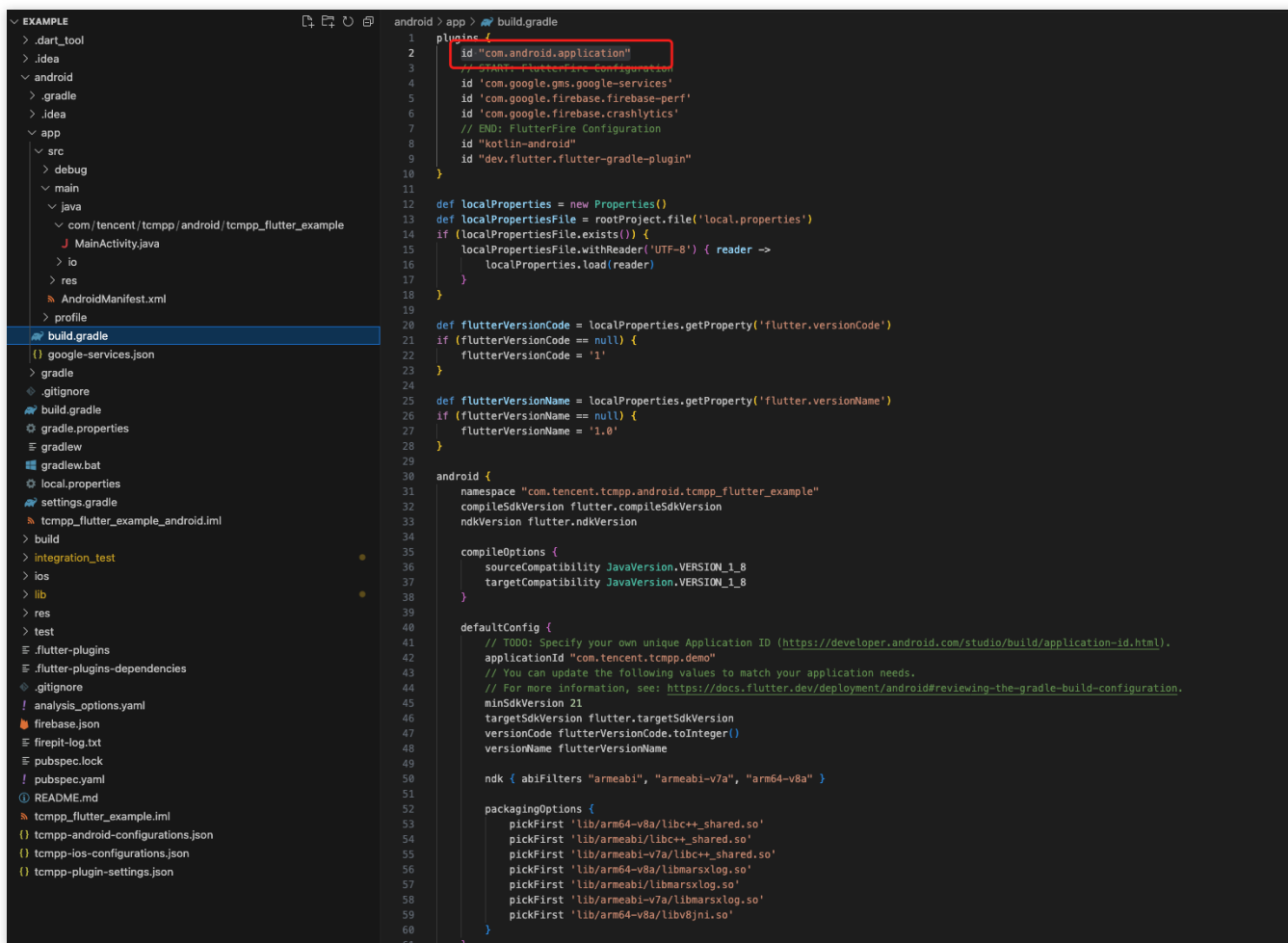
最近更新时间：2024-08-06 16:27:09

Firebase Flutter 插件一起使用产生崩溃的问题

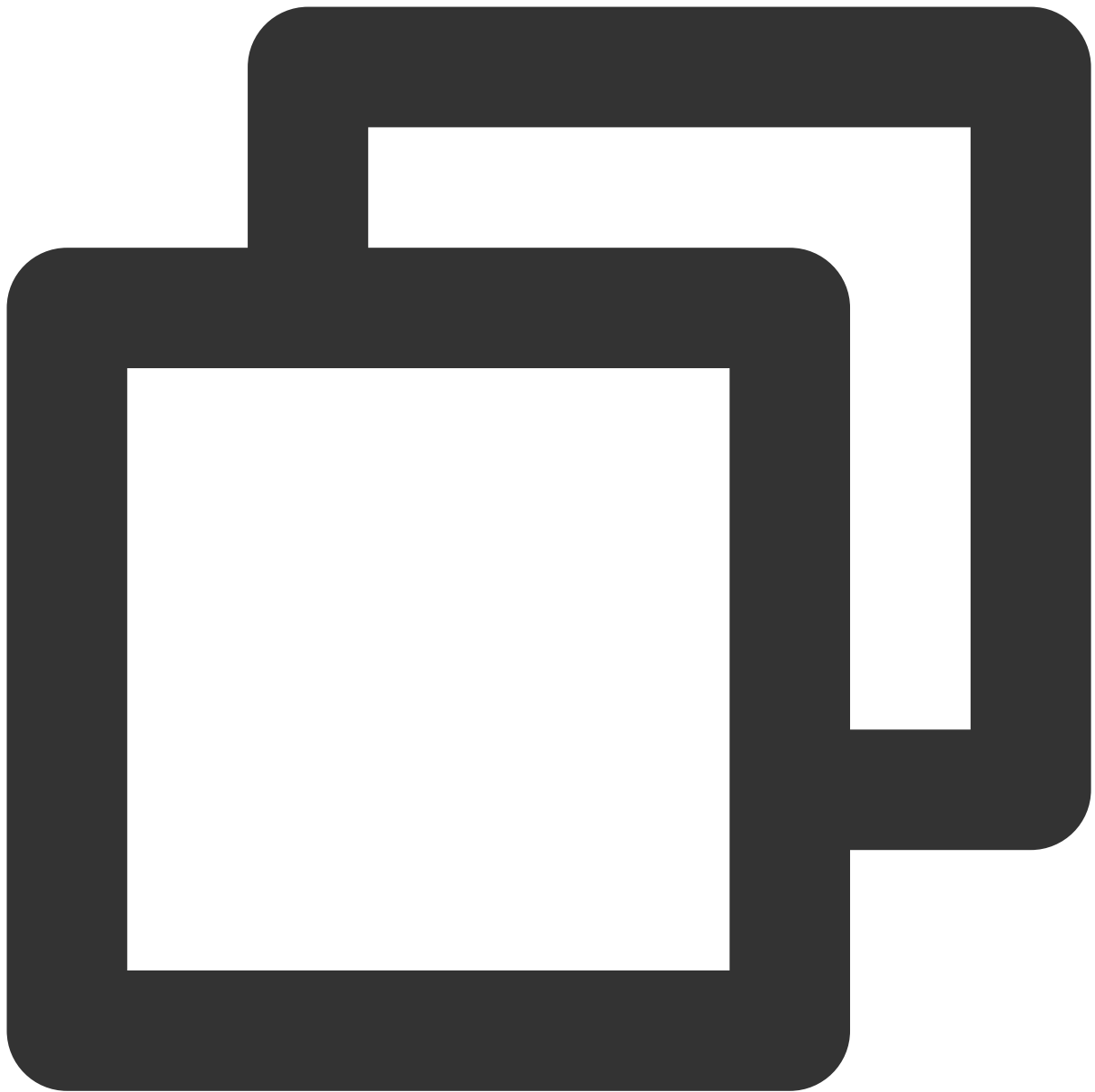
在 Android 平台上，Firebase Flutter 插件默认不支持多进程处理，因此可能会导致 TCMPP 小程序子进程崩溃。若要将 TCMPP Flutter 插件与 Firebase Flutter 插件一起使用，你需要自行在 TCMPP 小程序子进程中初始化 Firebase。

添加 TCMPP 依赖项

1. 找到您的 Flutter Android 项目根目录。在 Flutter 项目中，搜索 `id "com.android.application"` 包含此字符串的 `build.gradle` 文件所在的位置是您的 Flutter Android 项目根目录：



2. 在此 `build.gradle` 中的 `android` 闭包中添加依赖项：



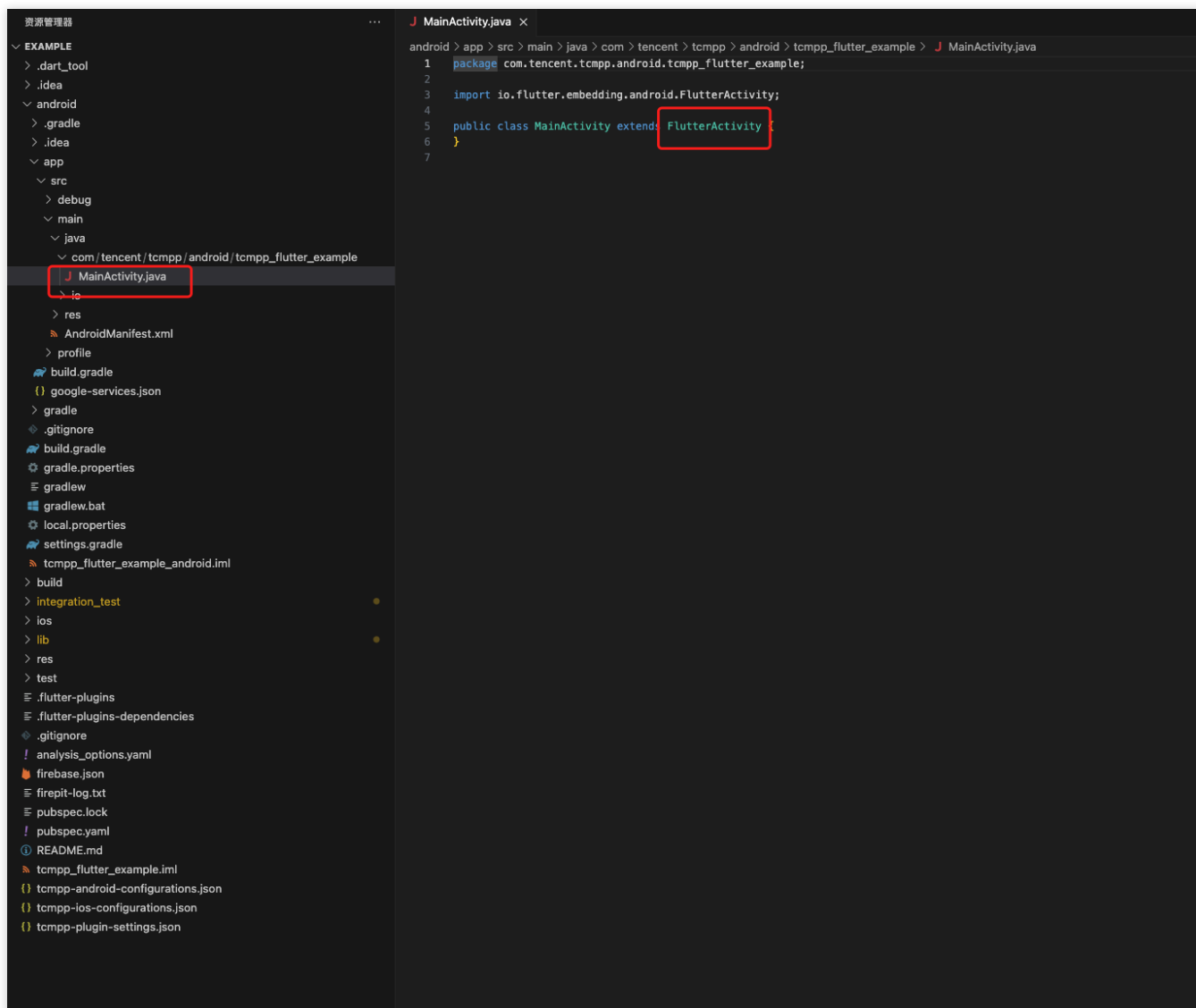
```
dependencies {  
    compileOnly 'com.tencent.tcmpp.android:mini_core:+'  
    compileOnly 'com.google.firebase:firebase-common:+'  
}
```

创建application类

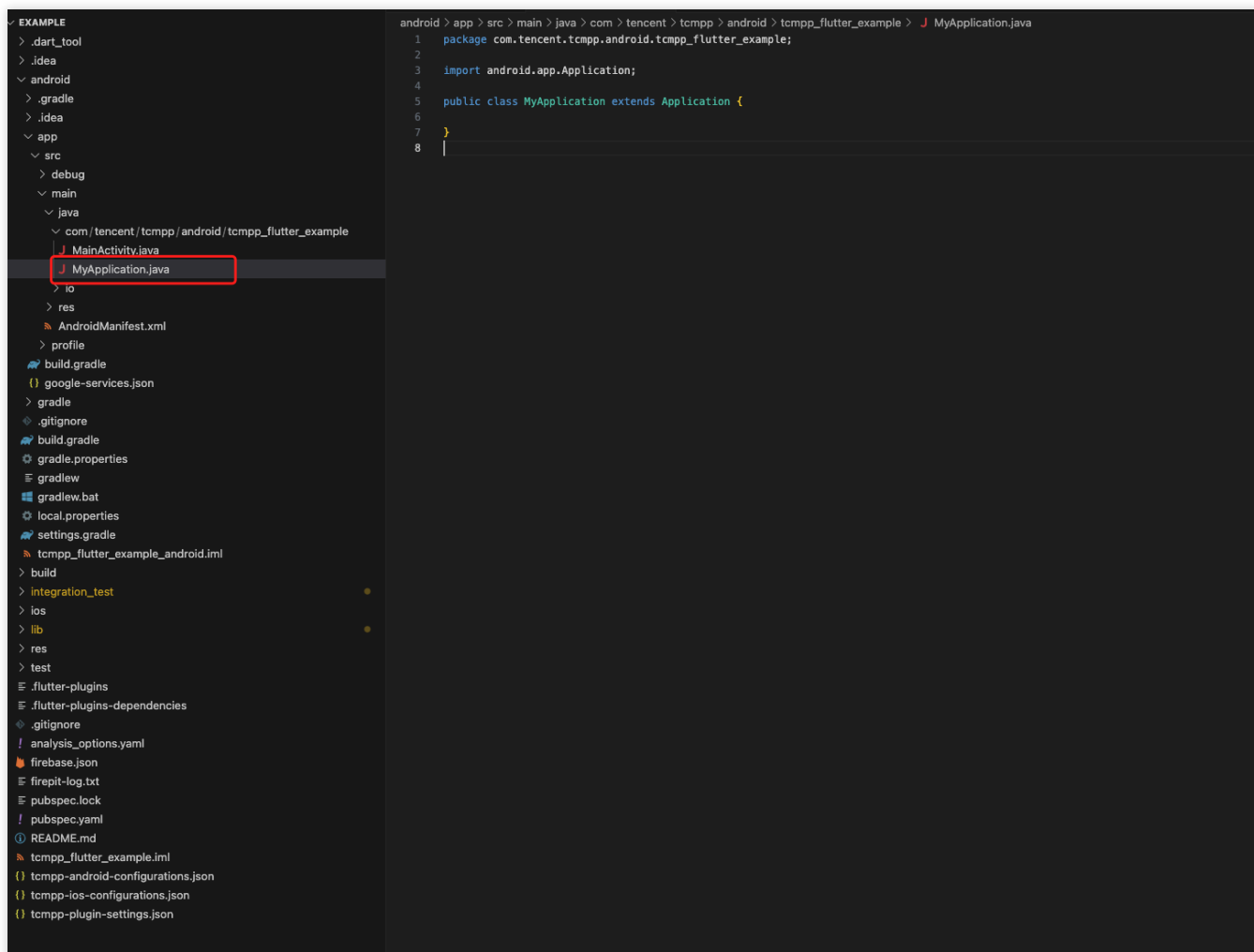
注意：

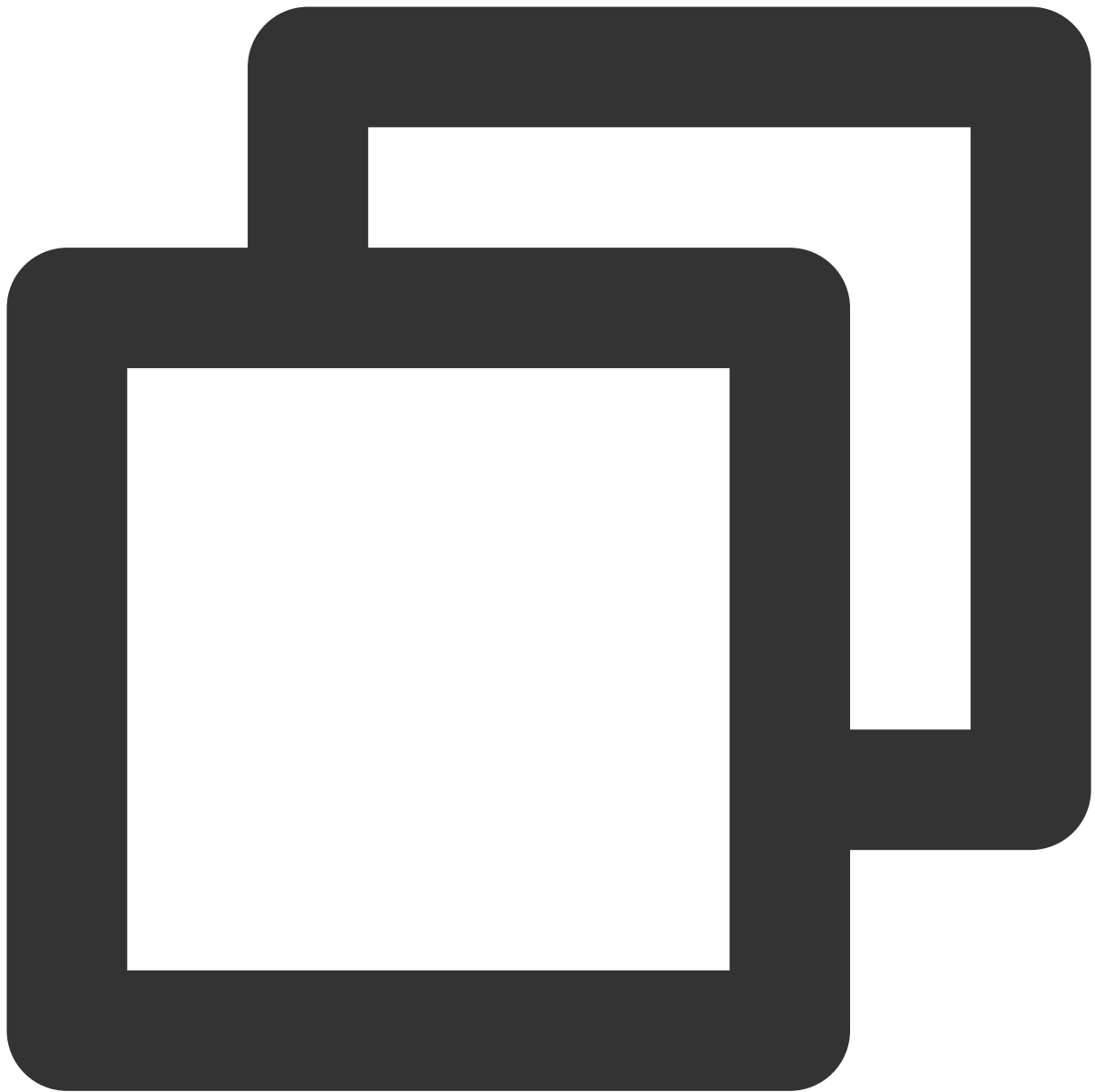
如果你的 Flutter 项目已经有自定义的 Android 应用程序类，请跳过此部分。

在 src 目录中，找到 FlutterActivity 所在的位置：



在 FlutterActivity 所在的目录中创建一个新的应用程序类。包与 FlutterActivity 的路径相同：





```
import android.app.Application;  
public class MyApplication extends Application {}
```

打开并编辑 `src > main` 下的 `AndroidManifest.xml`。在 `<application>` 标签中，更改 `android:name` 字段以匹配您的应用程序类：

EXAMPLE

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

android > app > src > main > AndroidManifest.xml > manifest

<manifest xmlns:android="http://schemas.android.com/apk/res/android">

<application

android:label="tcmp_flutter_example"

android:name="\${applicationName}"

android:extractNativeLibs="true"

android:icon="@mipmap/ic_launcher">

<activity

android:name=".MainActivity"

android:exported="true"

android:launchMode="singleTop"

android:theme="@style/LaunchTheme"

android:configChanges="orientation|keyboardHidden|keyboard|screenSize|smallestScreenSize|locale|layoutDirection|fontScale|screenLayout|density|uiMode"

android:hardwareAccelerated="true"

android:windowSoftInputMode="adjustResize">

<!-- Specifies an Android theme to apply to this Activity as soon as

the Android process has started. This theme is visible to the user

while the Flutter UI initializes. After that, this theme continues

to determine the Window background behind the Flutter UI. -->

<meta-data

android:name="io.flutter.embedding.android.NormalTheme"

android:resource="@style/NormalTheme"

/-->

<intent-filter>

<action android:name="android.intent.action.MAIN"/>

<category android:name="android.intent.category.LAUNCHER"/>

</intent-filter>

</activity>

<!-- Don't delete the meta-data below.

This is used by the Flutter tool to generate GeneratedPluginRegistrant.java -->

<meta-data

android:name="flutterEmbedding"

android:value="2" /-->

</application>

</manifest>

> .dart_tool

> .idea

> android

> .gradle

> .idea

> app

> src

> debug

> main

> java

> com/tencent/tcmp/flutter_example

> MainActivity.java

> MyApplication.java

> io

> res

> AndroidManifest.xml

> proguard

> build.gradle

> google-services.json

> gradle

> .gitignore

> build.gradle

> gradle.properties

> gradlew

> gradlew.bat

> local.properties

> settings.gradle

> tcmp_flutter_example_android.iml

> build

> integration_test

> ios

> lib

> res

> test

> flutter-plugins

> flutter-plugins-dependencies

> .gitignore

> analysis_options.yaml

> firebase.json

> firepit-log.txt

> pubspec.lock

> pubspec.yaml

> README.md

> tcmp_flutter_example.iml

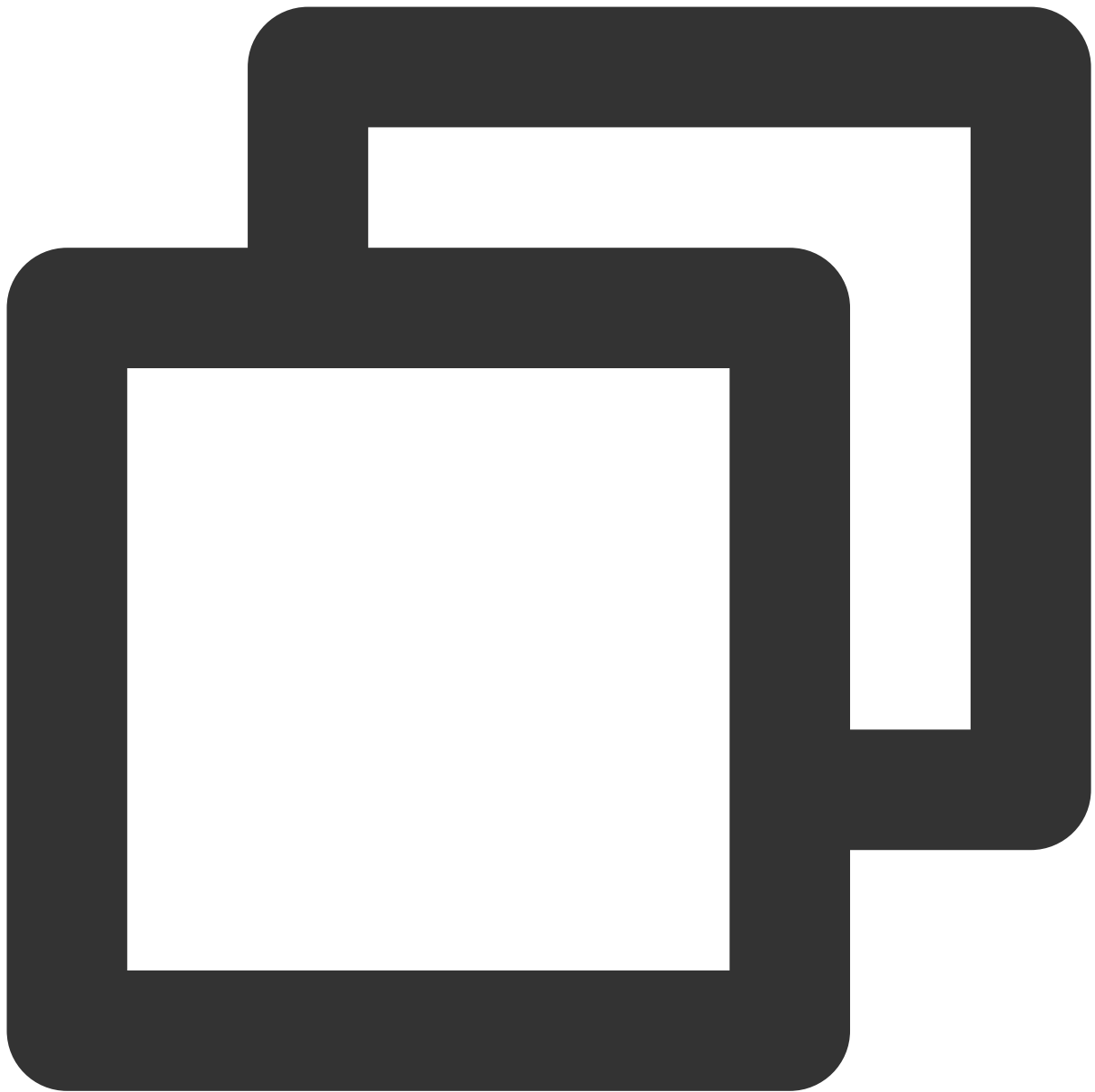
> tcmp-android-configurations.json

> tcmp-ios-configurations.json

> tcmp-plugin-settings.json

版权所有：腾讯云计算（北京）有限责任公司

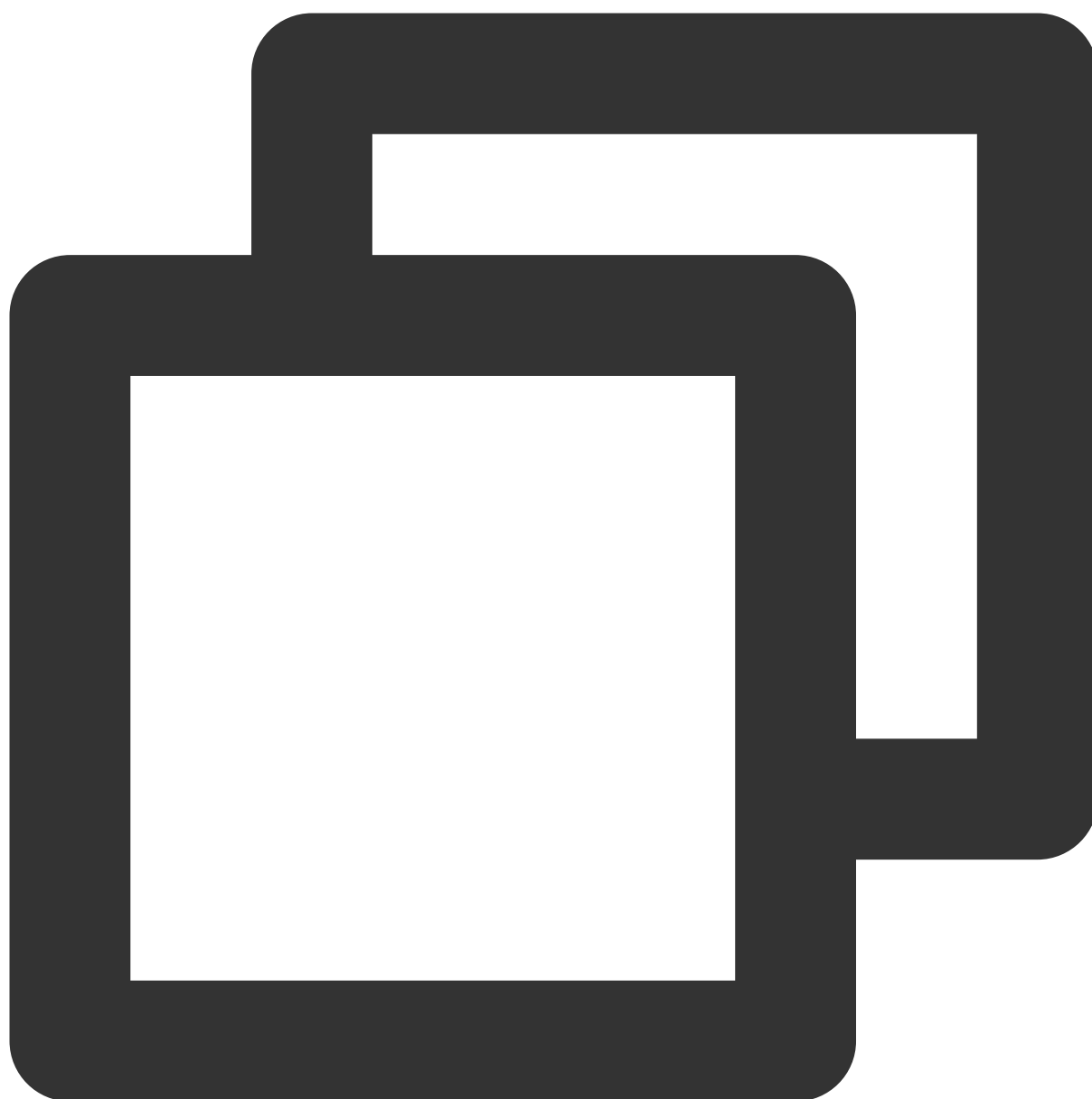
第573 共646页



```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"><application
```

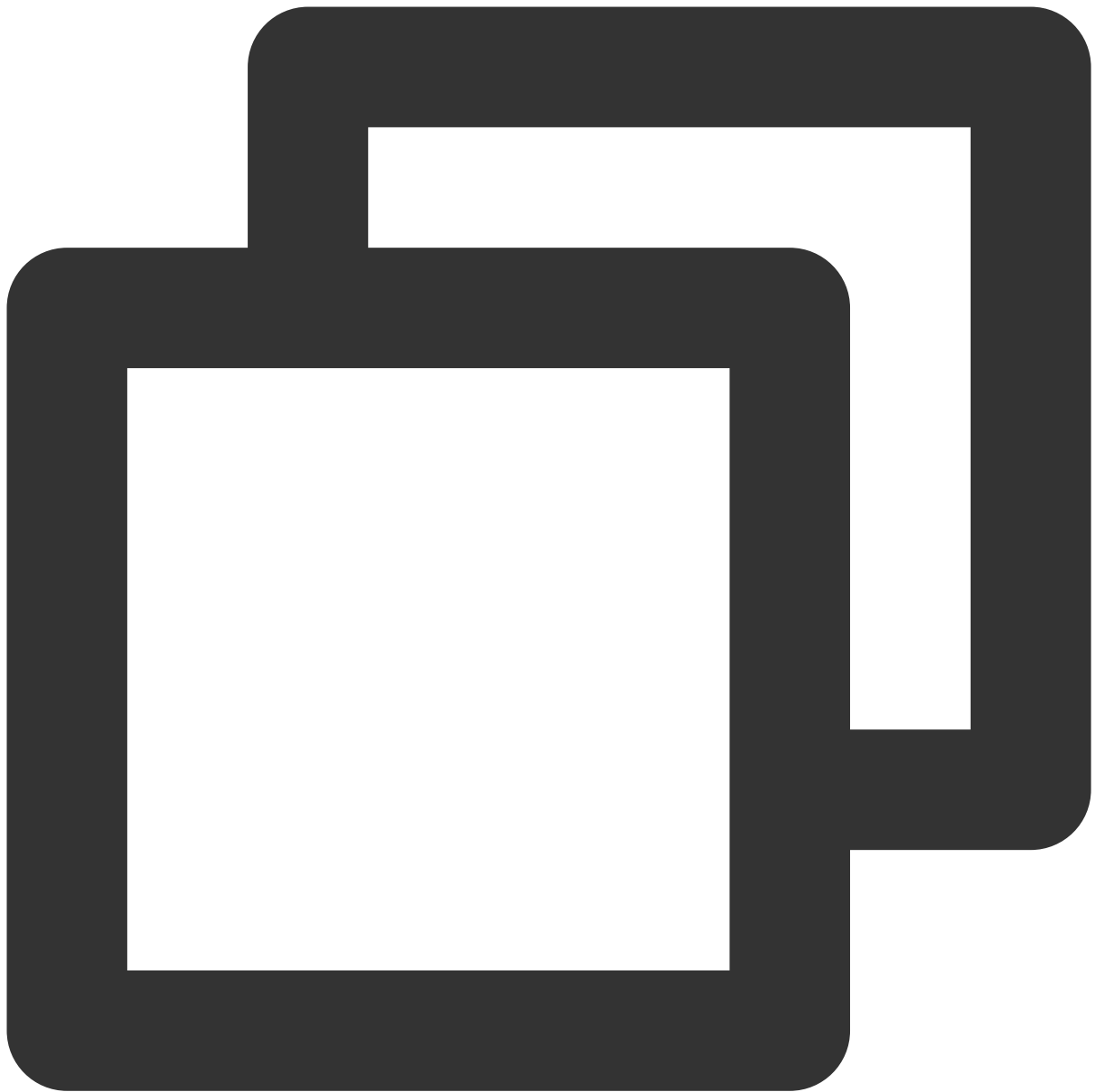
在子进程上初始化 **Firebase**

1. 在您的 **Android** 应用程序类中重写 **onCreate** 函数。如果您已经操作此步骤，请跳过。



```
public void onCreate() {  
    super.onCreate();  
}
```

2. 在此处添加子进程的 Firebase 初始化代码



```
import android.app.Application;
import com.google.firebase.FirebaseApp;
import com.tencent.tmf.mini.api.TmfMiniSDK;

public class MyApplication extends Application {

    @Override
    public void onCreate() {
        super.onCreate();

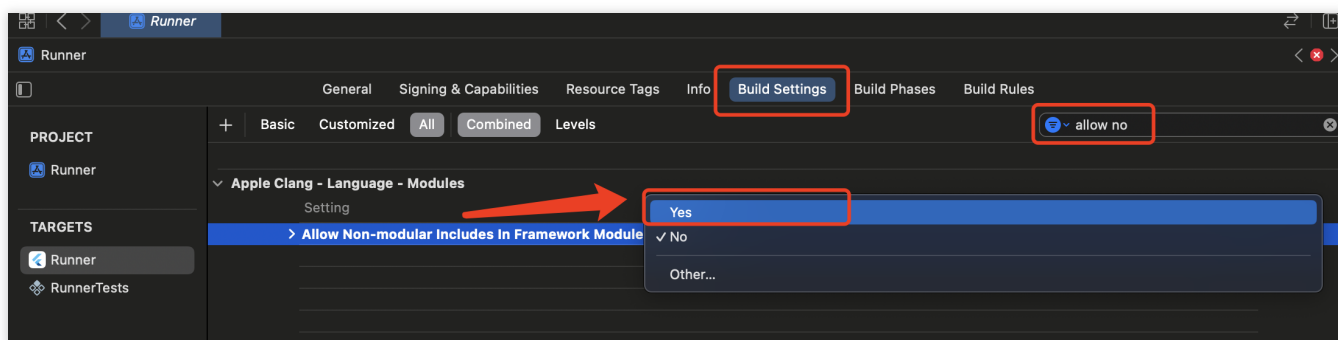
        ...
    }
}
```

```
if (TmfMiniSDK.isMiniProcess(this)) {  
    FirebaseApp.initializeApp(this);  
}  
  
...  
}  
}
```

iOS端编译产生找不到头文件的报错

```
1 //  
2 // Generated file. Do not edit.  
3 //  
4 //  
5 // clang-format off  
6 //  
7 #import "GeneratedPluginRegistrant.h"  
8 //  
9 #if __has_include(<tcmpp_flutter/TcmppFlutterPlugin.h>)  
10 #import <tcmpp_flutter/TcmppFlutterPlugin.h>  
11 #else  
12 @import tcmpp_flutter;  
13 #endif  
14 //  
15 @implementation GeneratedPluginRegistrant  
16 //  
17 + (void)registerWithRegistry:(NSObject<FlutterPluginRegistry>*)registry {  
18     [TcmppFlutterPlugin registerWithRegistrar:[registry registrarForPlugin:@"TcmppFlutterPlugin"]];  
19 }  
20 //  
21 @end  
22
```

使用 Xcode 打开工程，选择正确的 target->Build Settings。搜索框中输入"Allow Non-modular Includes In Framework Modules"，然后将其设置为 "Yes"：



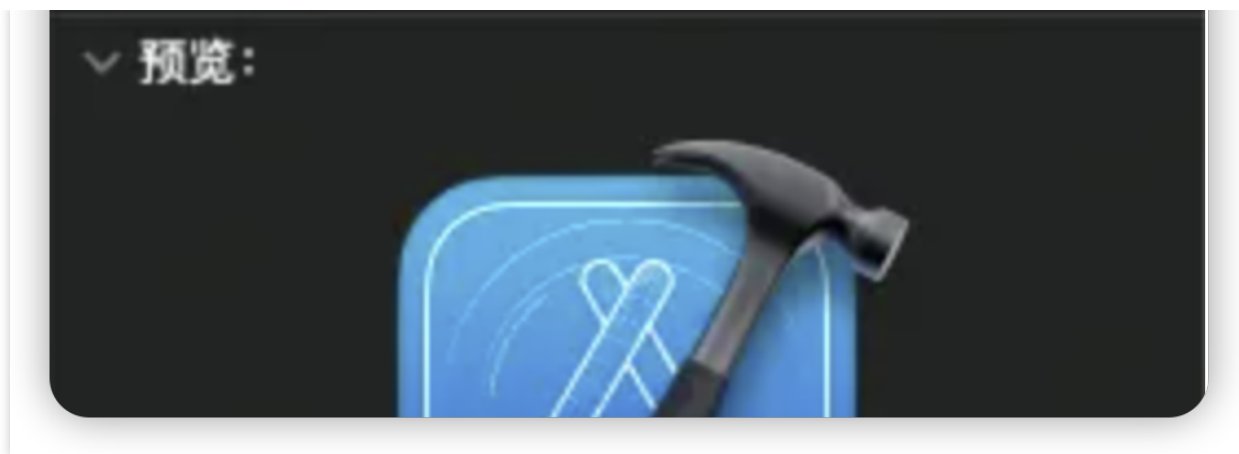
Apple M系列芯片电脑模拟器运行异常

当前 TCMPP SDK 版本对于M系列芯片电脑模拟器上运行需要通过 Rosetta 运行。

xcode14之前：

我们可以右键 xcode > 显示简介 > 勾选使用 Rosetta 打开，即可在模拟器上运行：

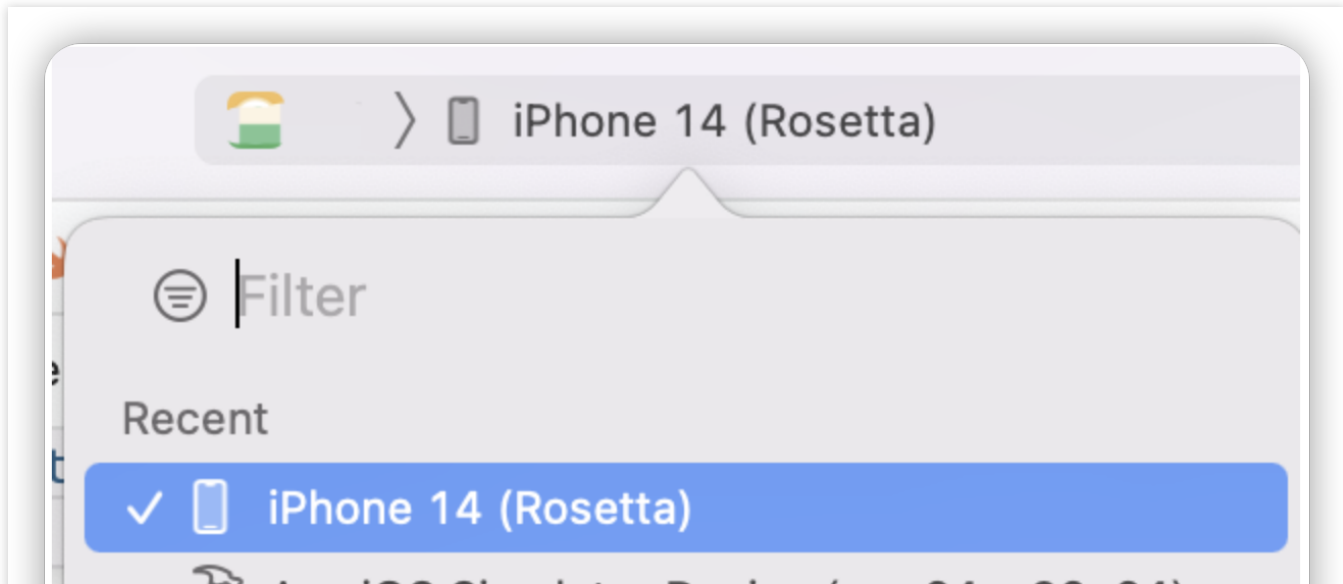
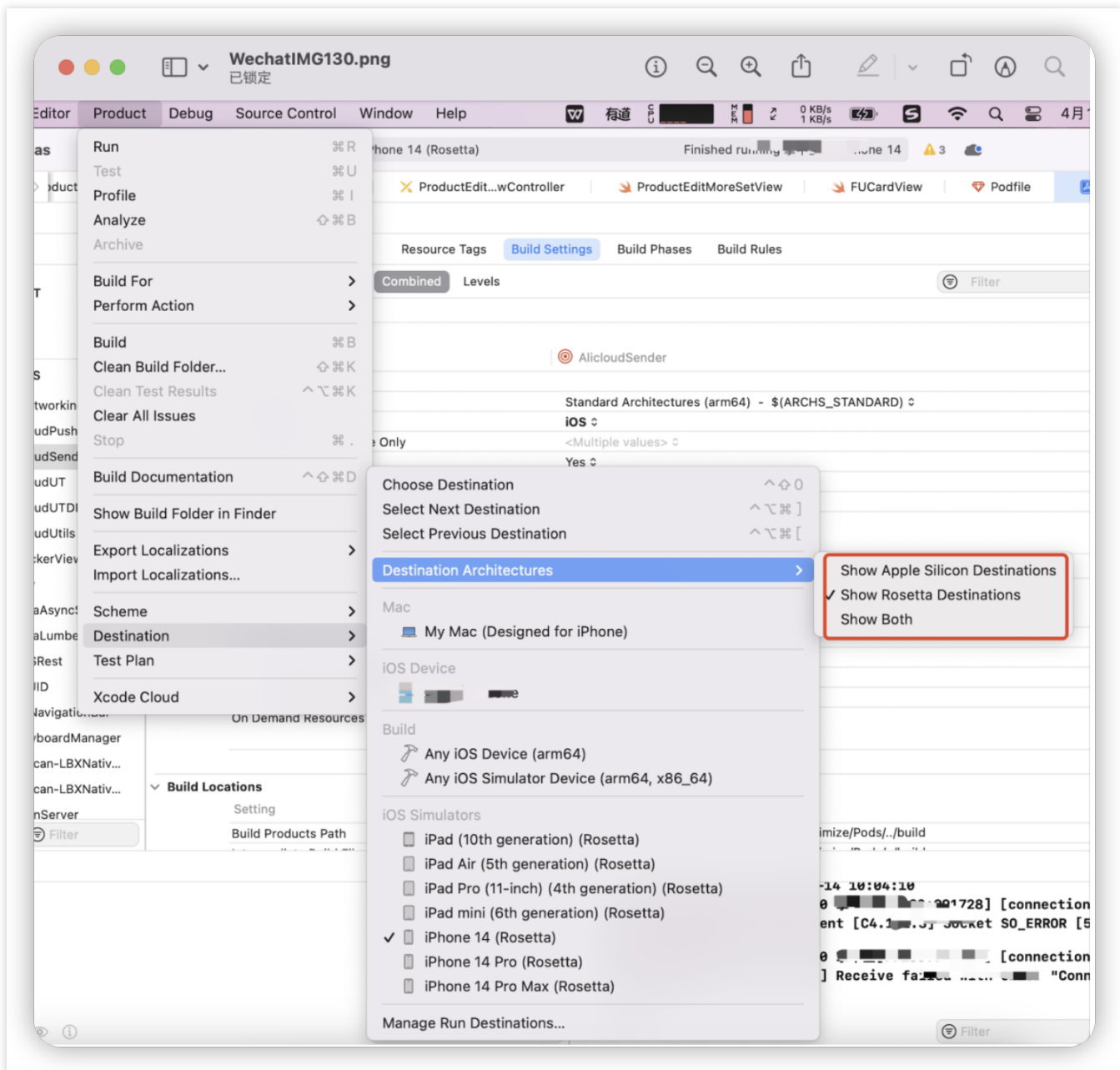


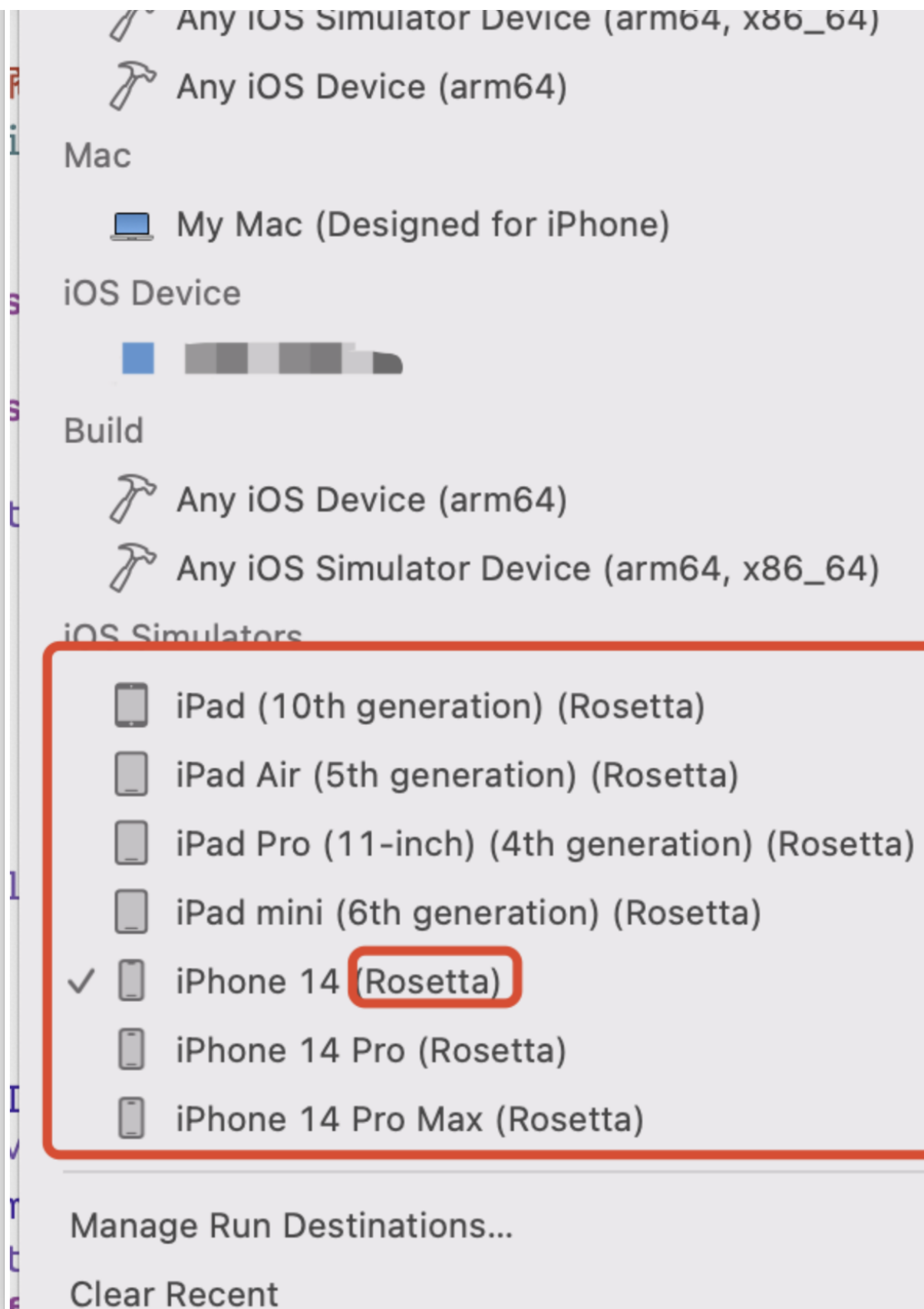


xcode14后：

xcode打开项目 > Product > Destination > Destination Architecturesk 可以选择用哪种模式的模拟器打开。

我们选择 （Rosetta）结尾的模拟器即可：





Flutter API

小程序管理 API

打开小程序

最近更新时间：2024-08-06 16:01:23

打开小程序

打开小程序时，会先判断本地是否有缓存的小程序，如果没有，则会自动从远程服务器上下载小程序，然后打开。
如果有缓存的小程序，则会先打开本地小程序，同时后台校验服务器端是否有新版本。

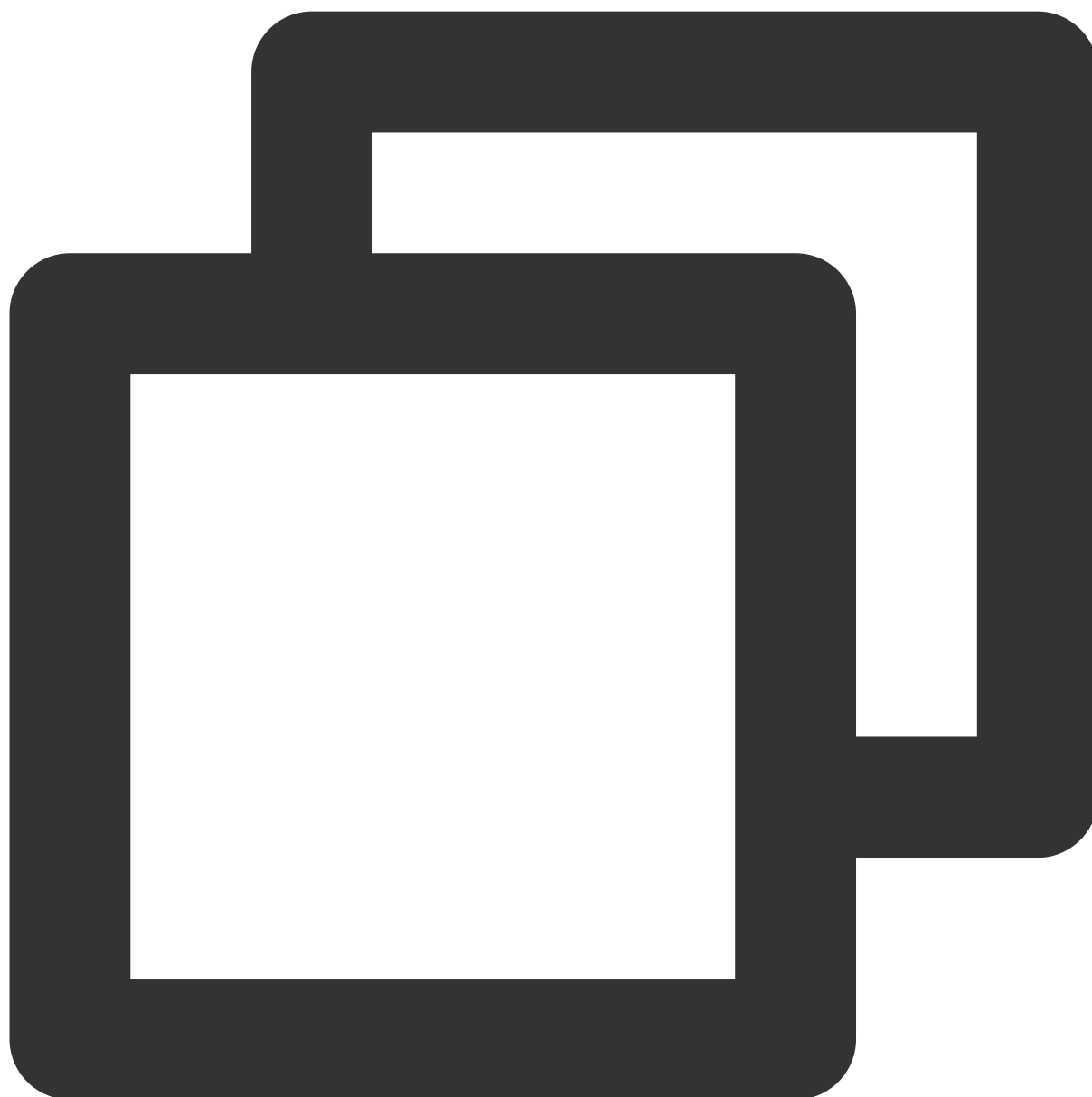
说明：

如果有新版本，则下载新版小程序，下次打开时，就会使用新版小程序。

使用给定的 `appId` 启动小程序：

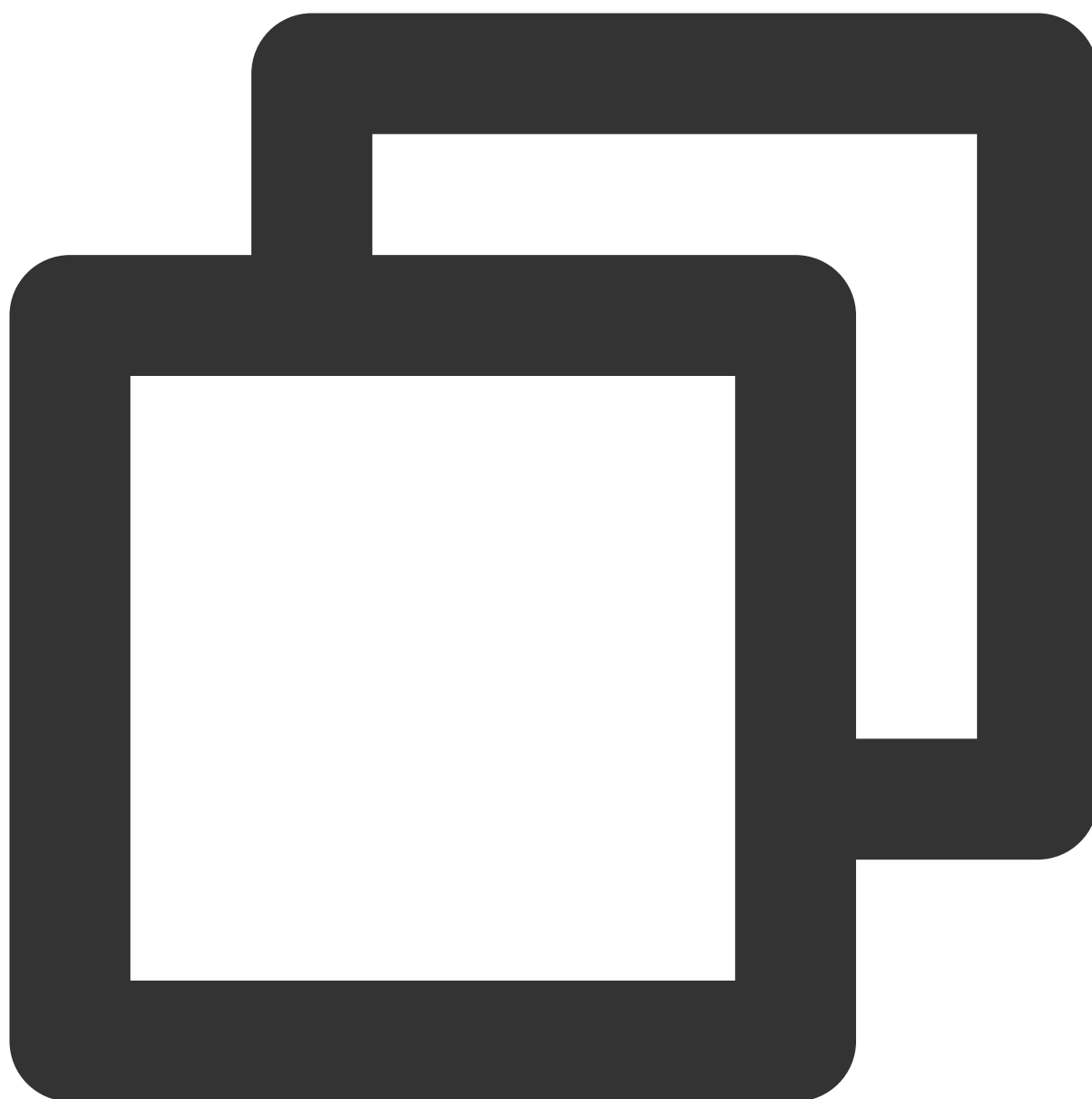
`appId`：小程序 ID

选项：此小程序的启动选项



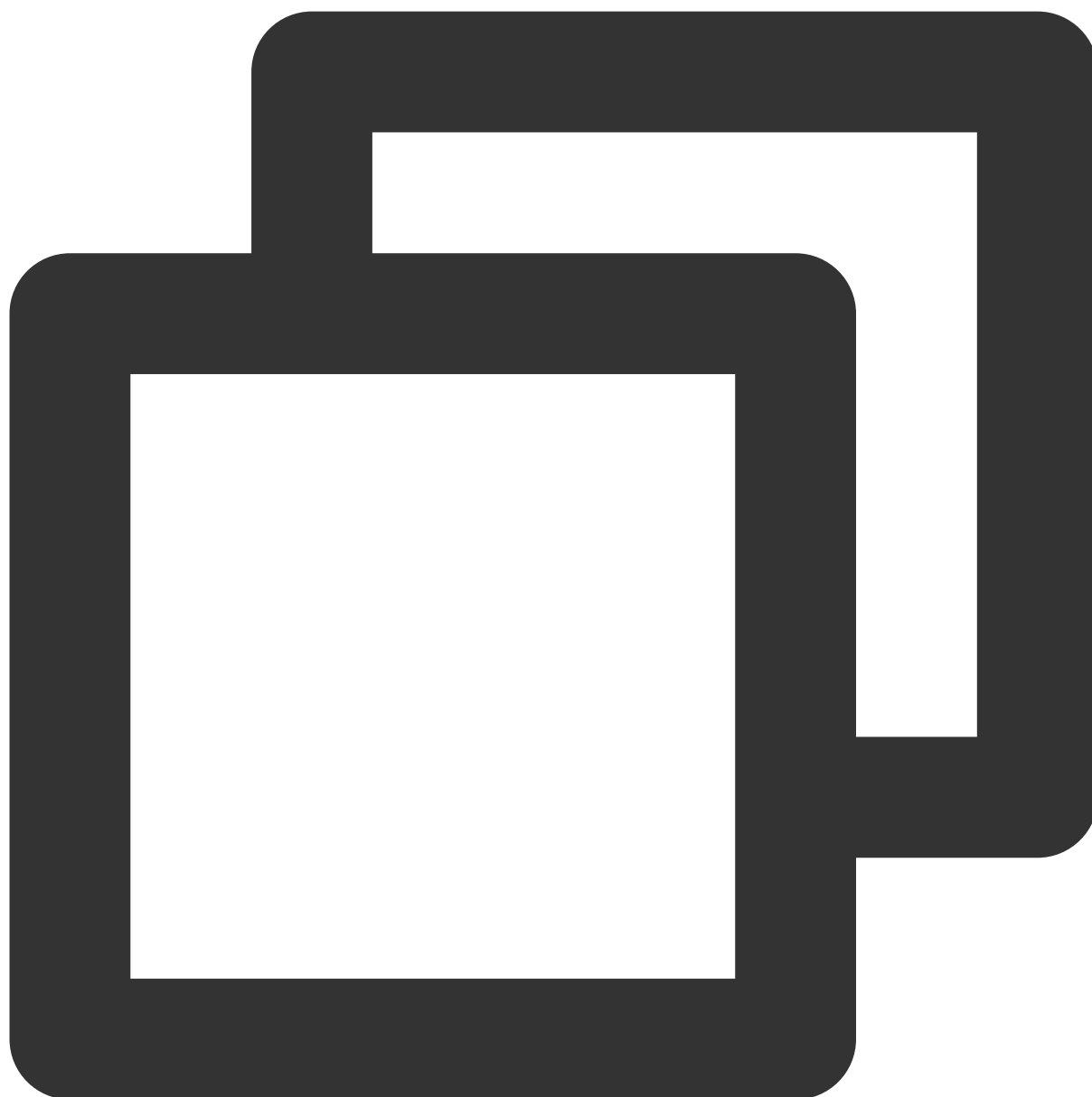
```
Future<void> startMiniAppWithId(String appId, MiniStartOptions? options)
```

示例代码：



```
_tcmppFlutterPlugin.startMiniAppWithId(appId!, null);
```

启动选项：

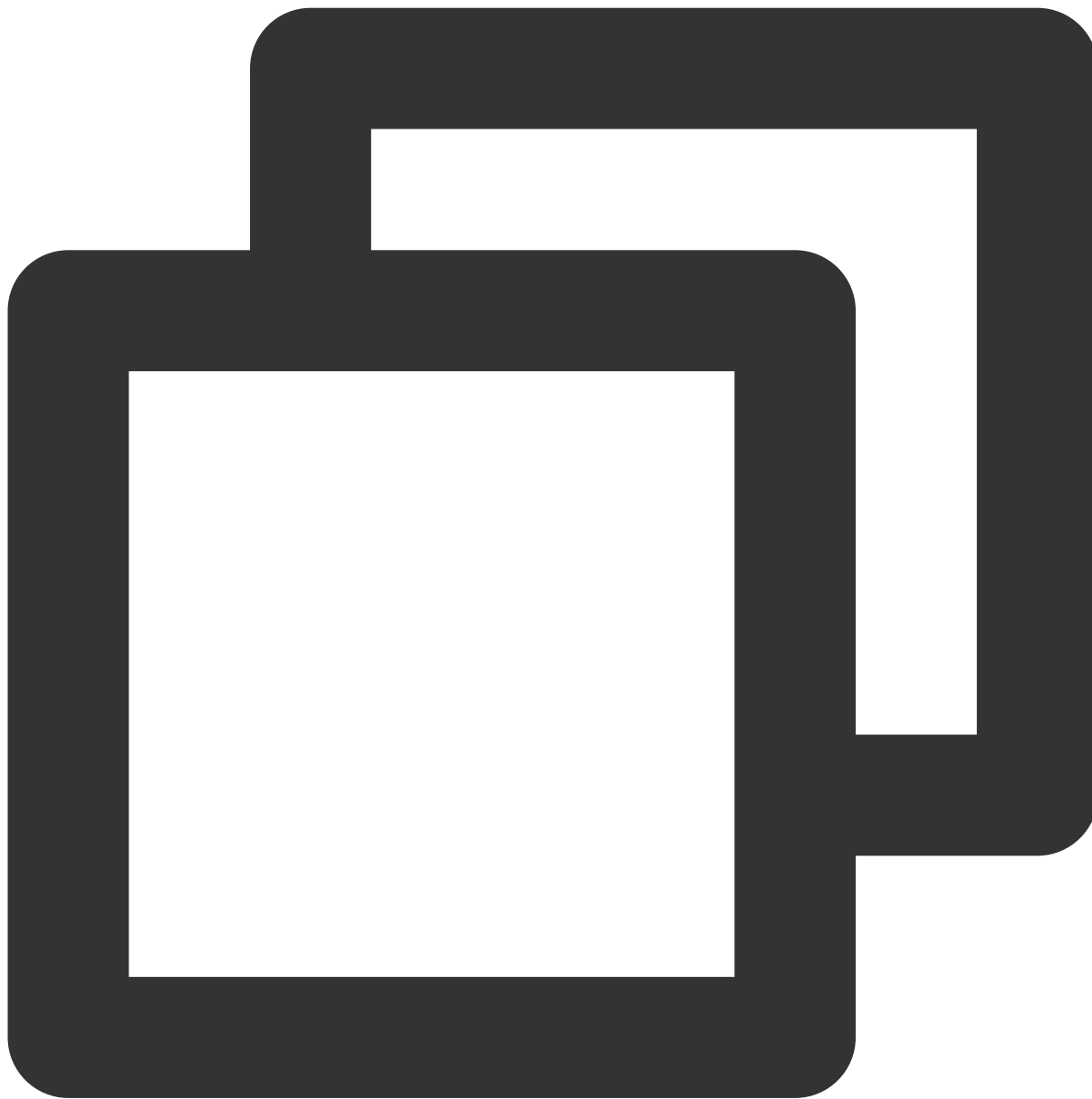


```
class MiniStartOptions {  
    /// 小程序的进入路径  
    String? entryPath;  
  
    /// 启动时始终更新小程序  
    bool? isForceUpdate;  
  
    /// 启动时传递给小程序的字符串参数  
    String? params;  
}
```

使用给定链接启动小程序：

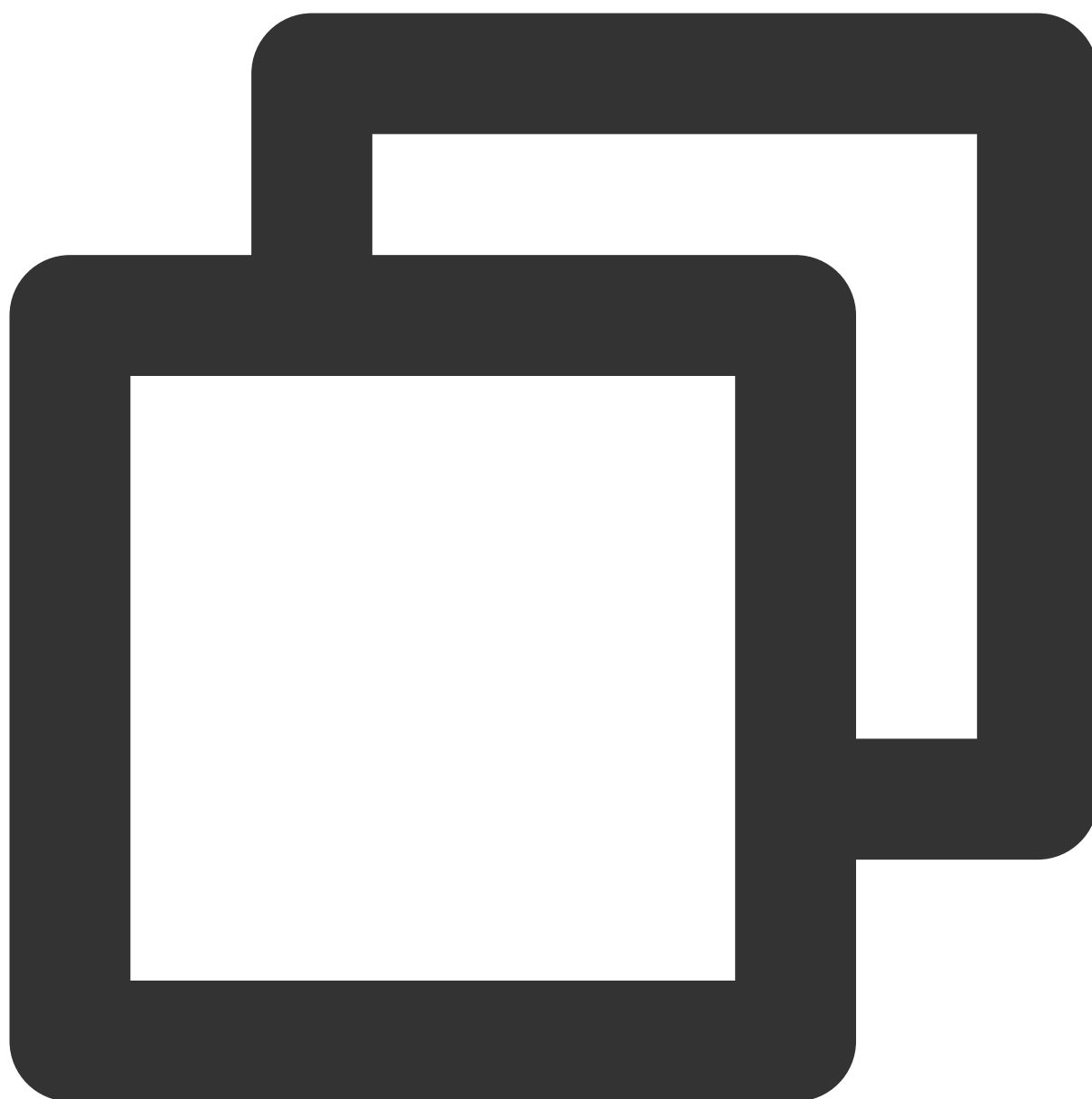
链接：要启动的 uri 链接

options：该小程序的启动选项



```
Future<void> startMiniAppWithLink(String link, MiniStartOptions? options)
```

示例代码：



```
_tcmppFlutterPlugin.startMiniAppWithLink(appId!, null);
```

关闭小程序

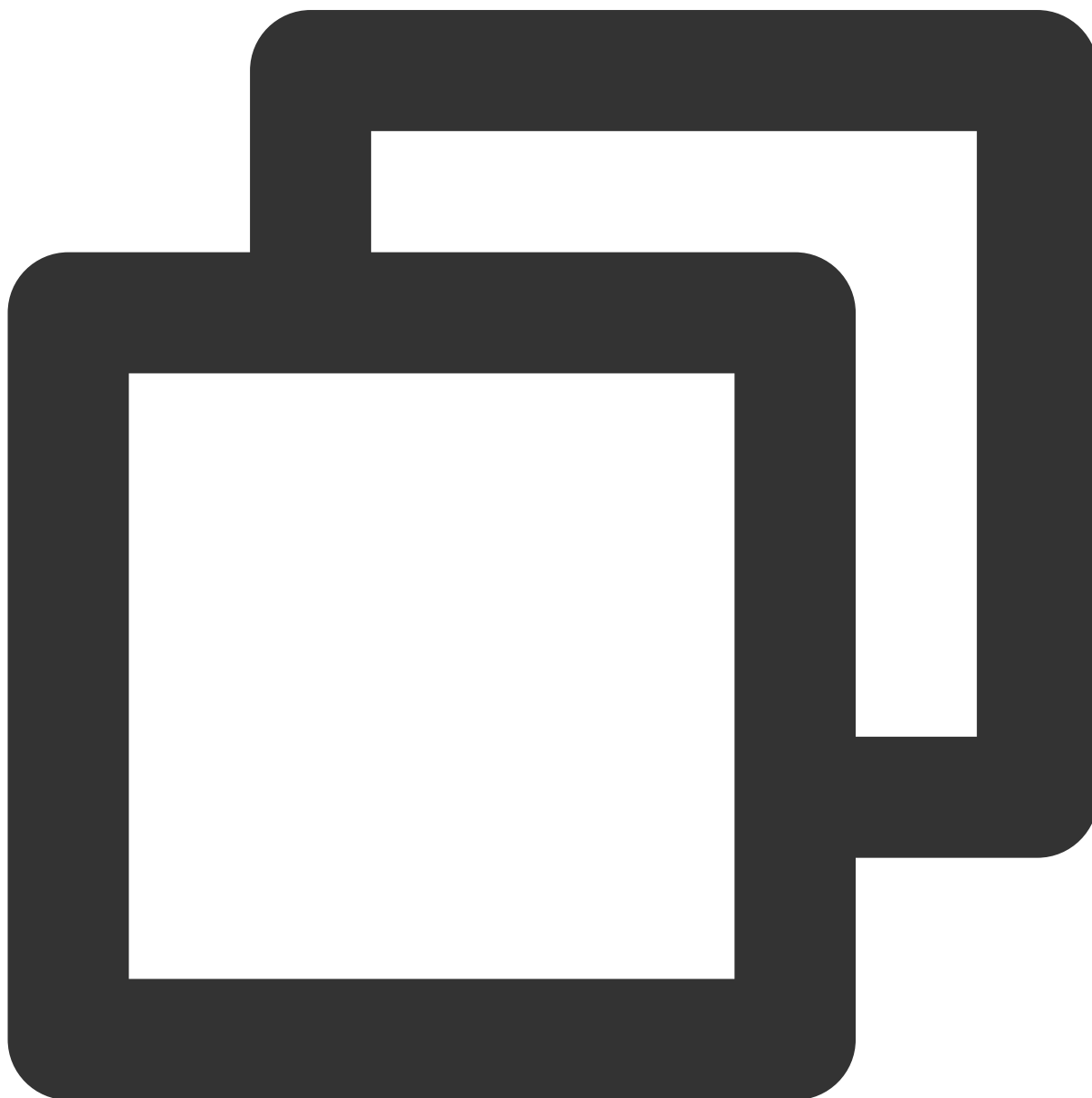
最近更新时间：2024-08-06 15:59:20

说明：

关闭小程序会在内存中清除小程序，下次打开时，需要重新将小程序加载到内存，耗时较多。

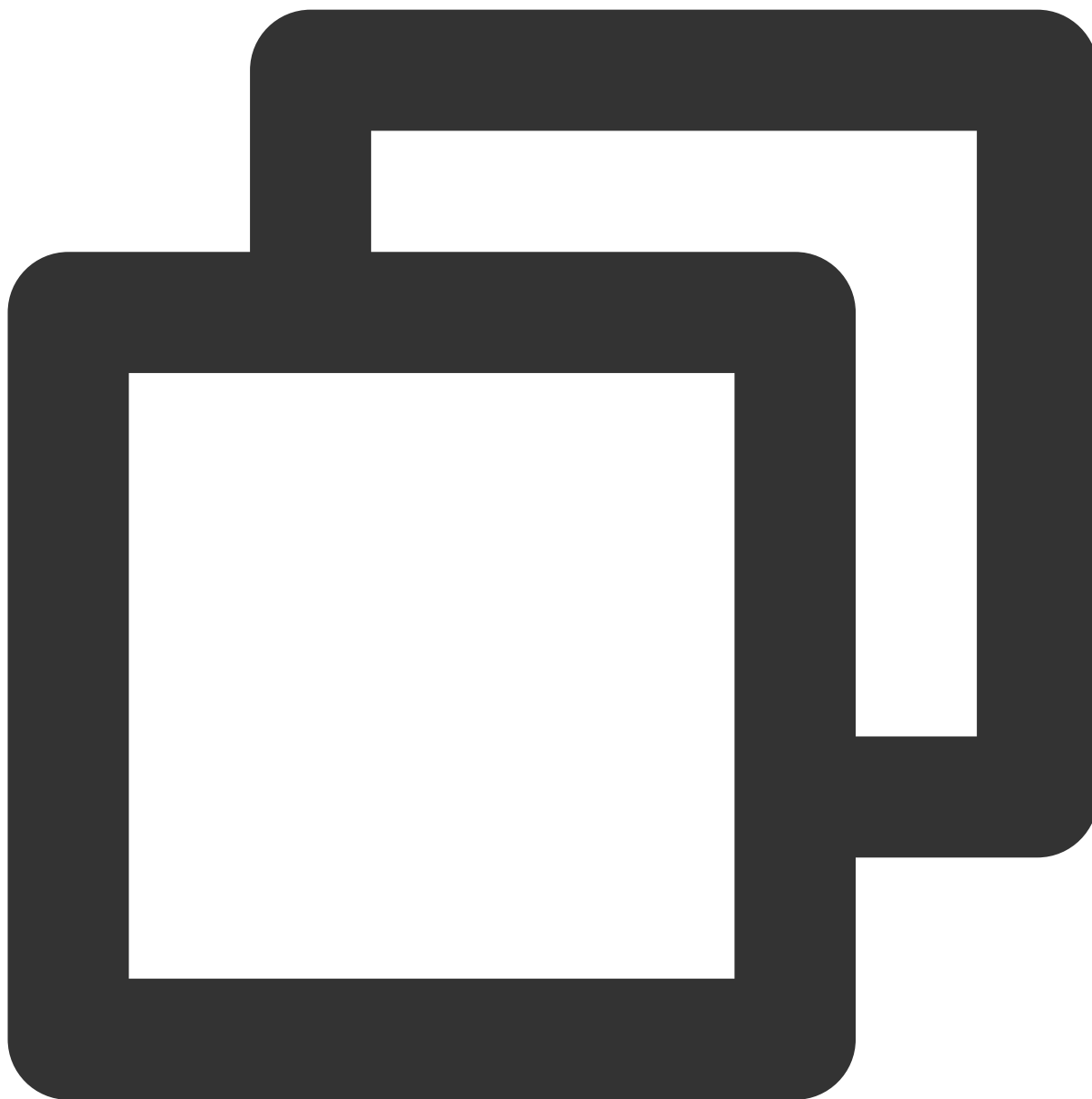
使用给定的应用程序 ID 停止小程序：

appId：小程序的应用程序 ID。



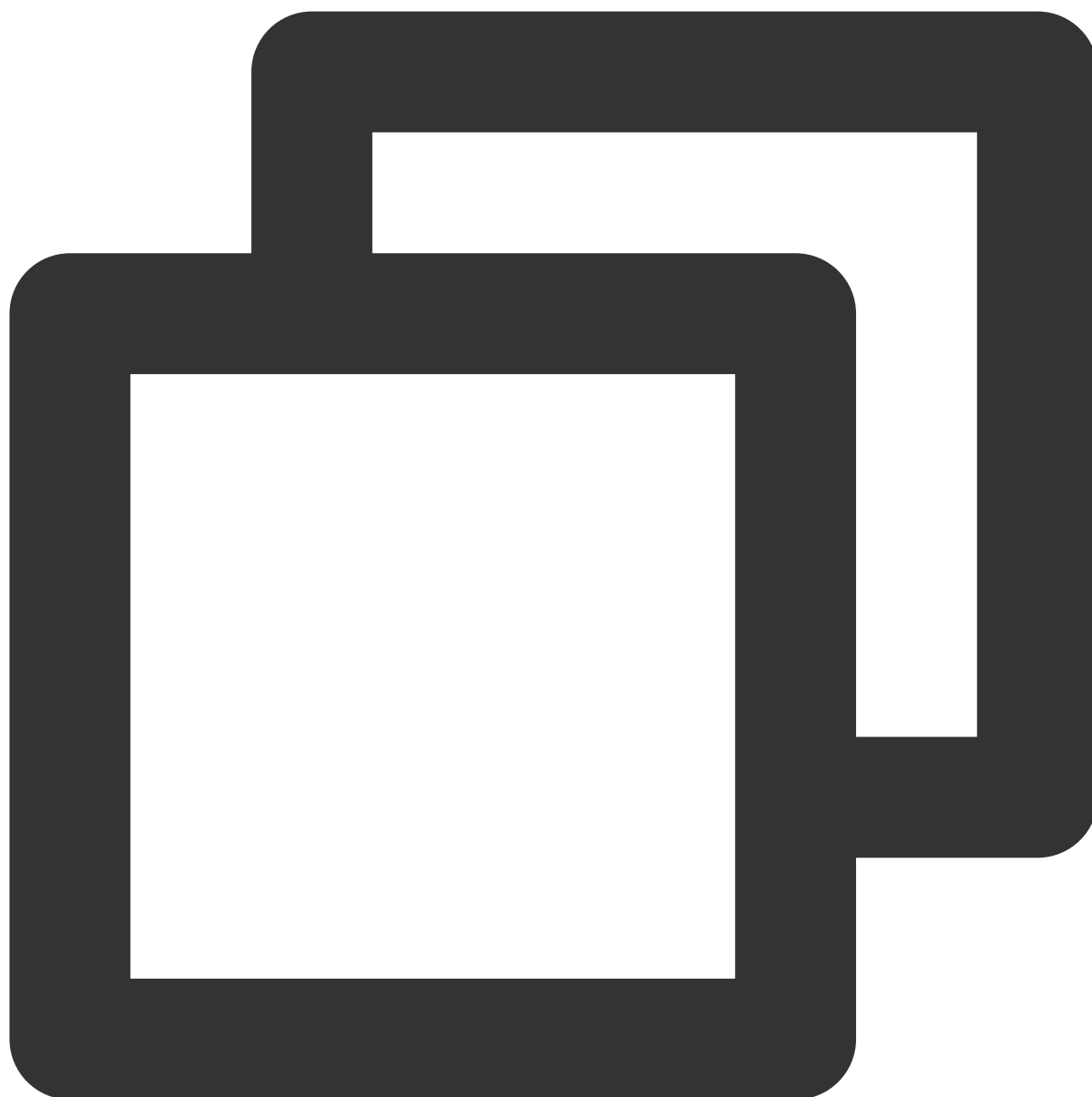
```
Future<void> stopMiniApp(String appId)
```


示例代码：



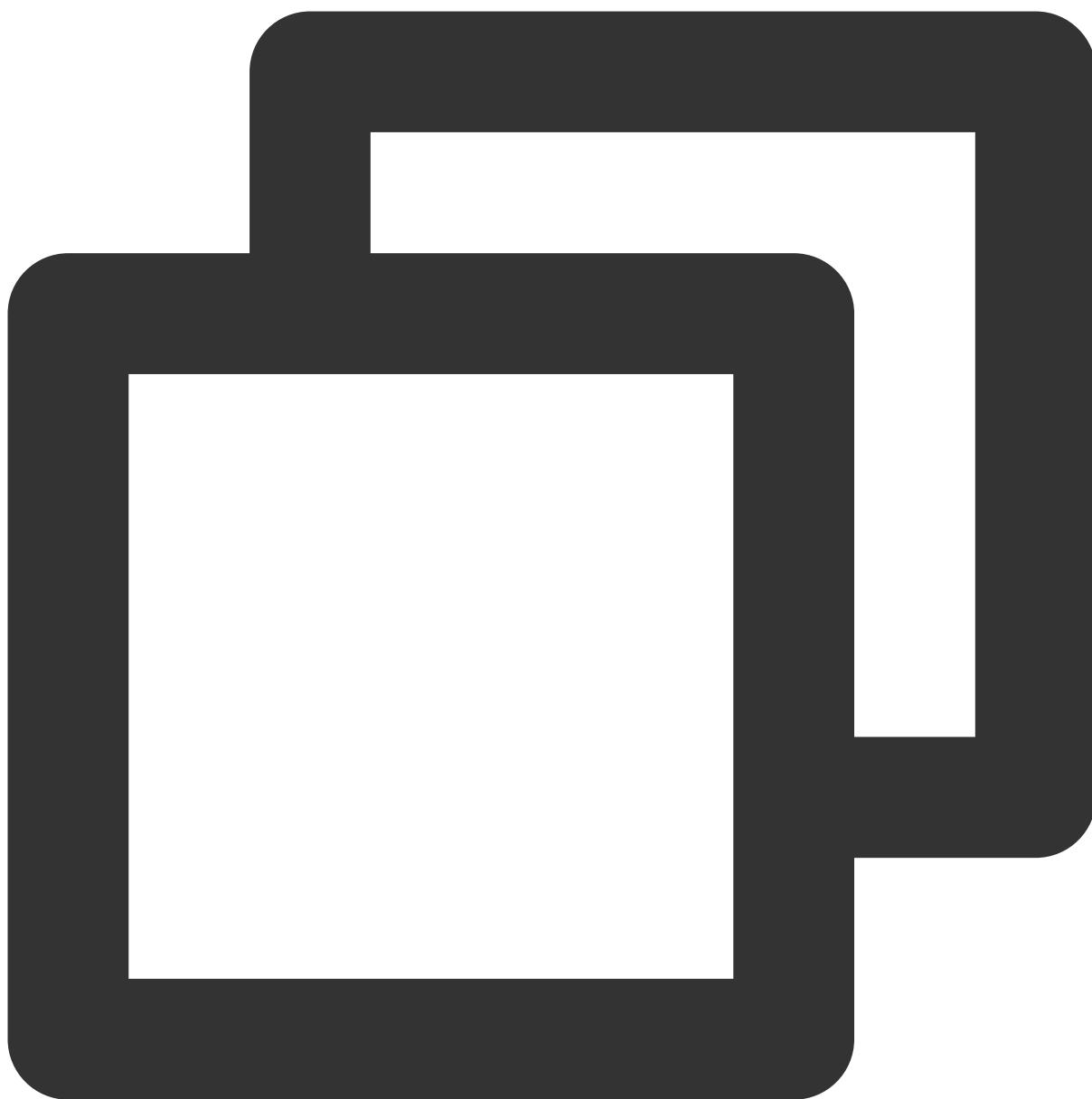
```
_tcmppFlutterPlugin.stopMiniApp(appId!);
```

停止所有正在运行的小程序：



```
Future<void> stopAllMiniApp()
```

示例代码：



```
_tcmppFlutterPlugin.stopAllMiniApp();
```

删除小程序

最近更新时间：2024-08-06 15:58:25

说明：

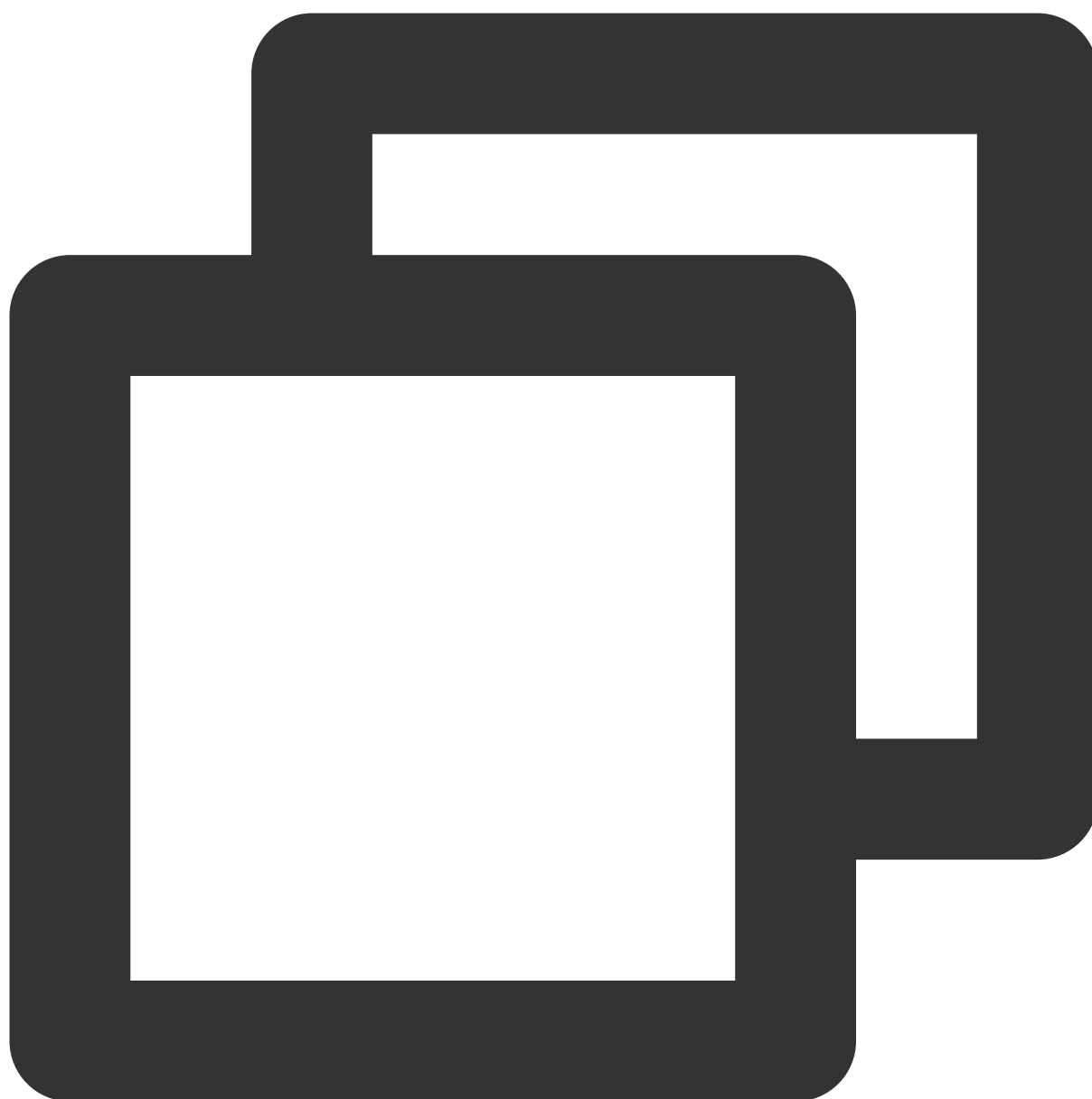
由于小程序的运行，会将小程序包和小程序信息缓存在本地，以便下次更快速的打开。如果想要将小程序的所有信息都删除，可以调用以下 **API** 删除某个小程序或者删除所有小程序。

删除给定 ID 的小程序的所有本地数据：

appId：小程序 ID

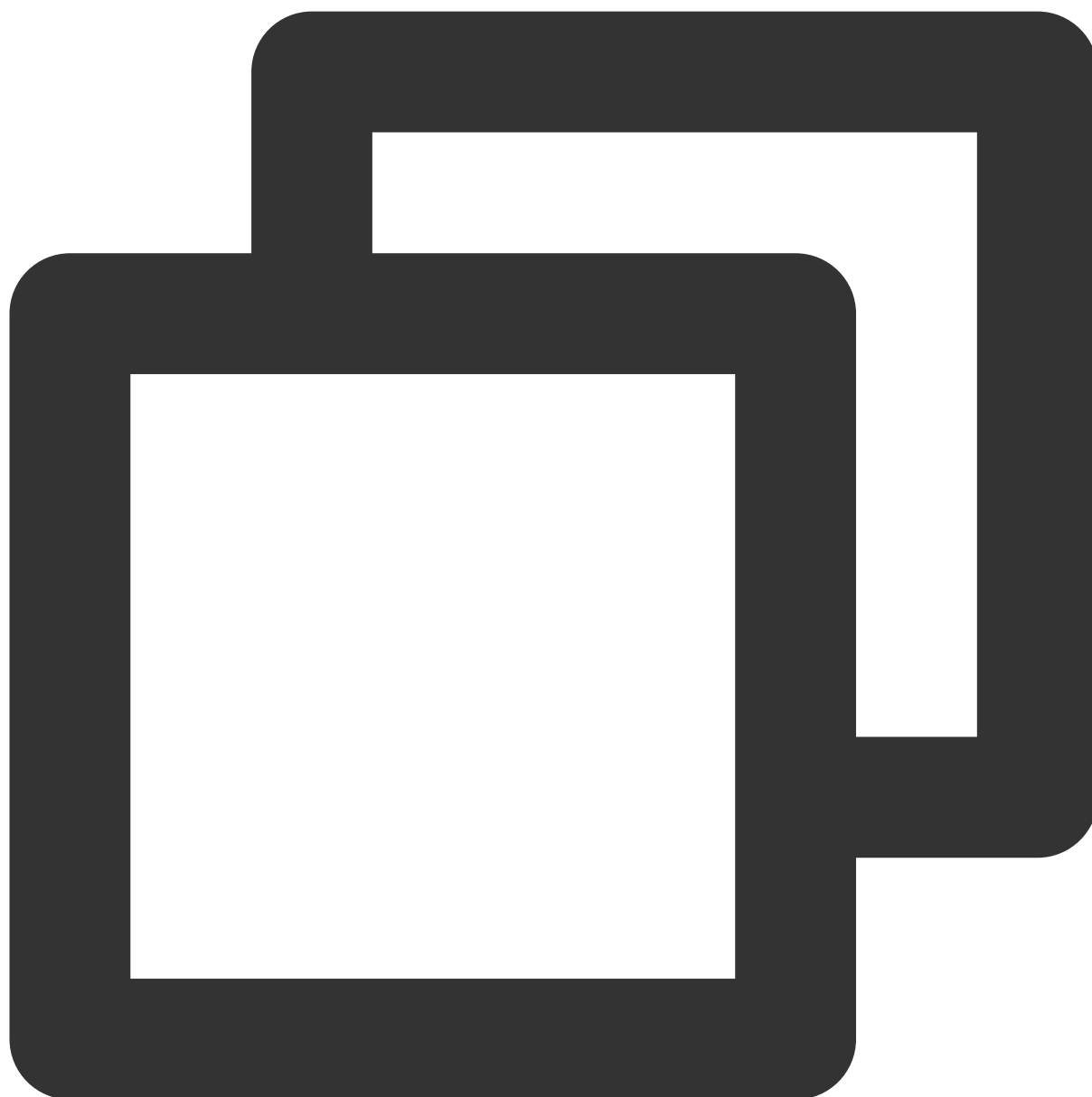
appVerType：小程序类型

version：要删除的小程序版本



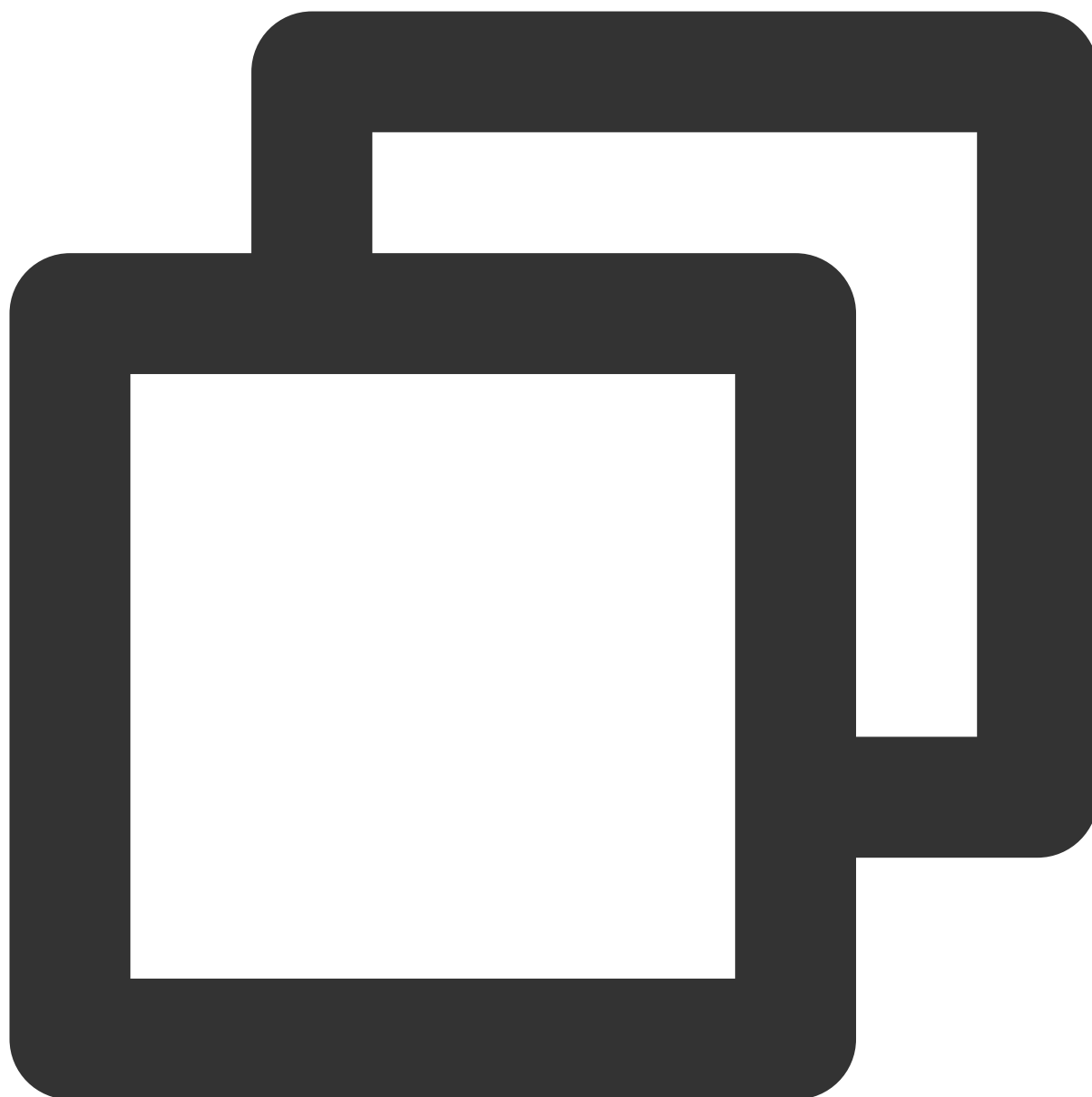
```
Future<void> deleteMiniAppCache(String appId,  
    {int appVerType = 0, String? version})
```

appVerType



```
/// 小程序软件包类型的常数
class AppVerType {
    static const int online = 0;
    static const int develop = 1;
    static const int preview = 2;
    static const int experience = 3;
}
```

示例代码：



```
_tcmppFlutterPlugin.deleteMiniAppCache (appId!, 0, '1.0.0') ;
```

搜索小程序

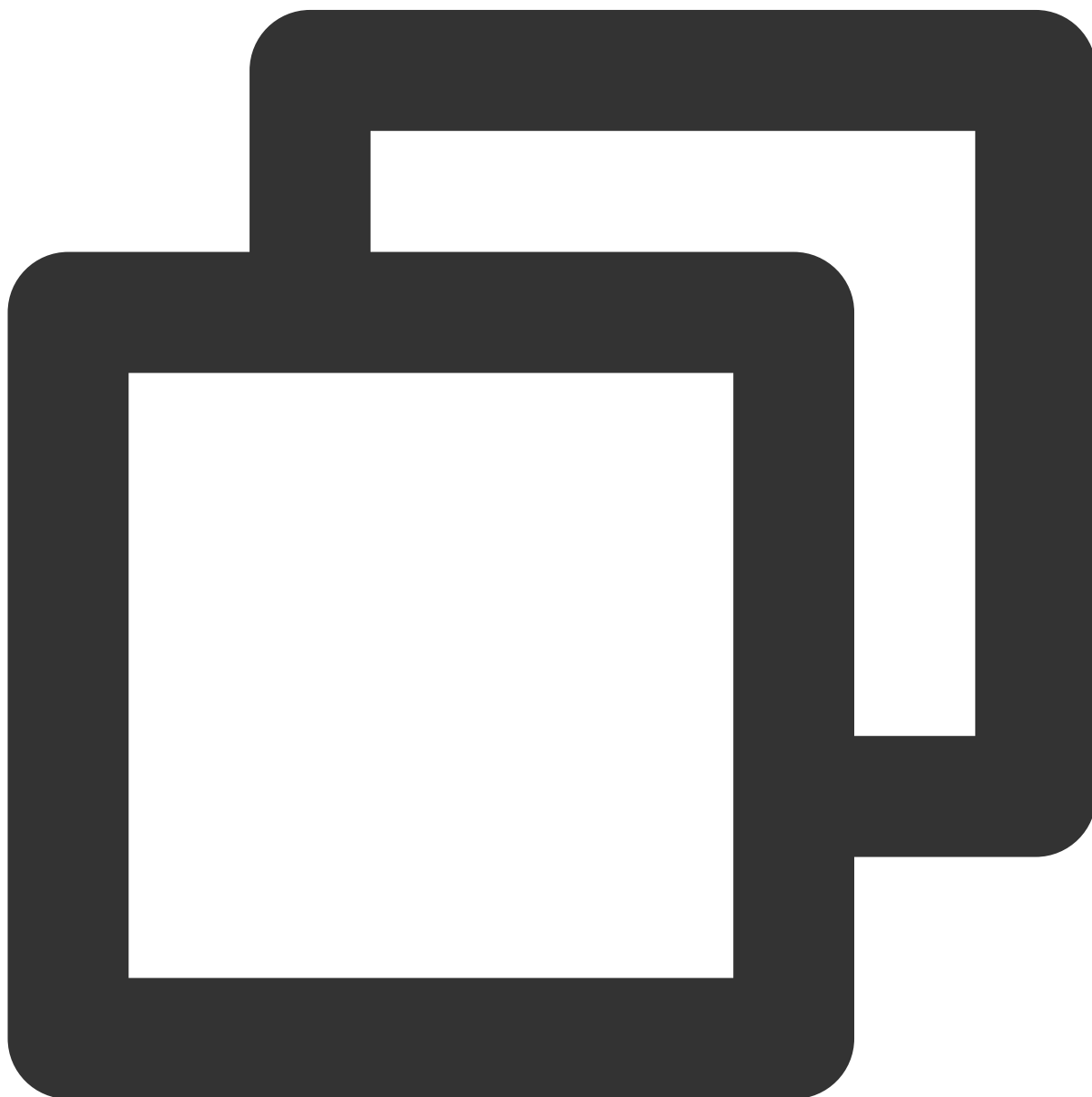
最近更新时间：2024-08-06 15:55:30

用给定的关键字搜索在线小程序：

keyWord：要搜索的小程序的关键字。

pageIndex：如果 **pageSize** 的值不为零，即启用了分页功能，则返回的页面（仅对 **Android** 生效）。

pageSize：页面大小：服务器返回页面的最大尺寸。设置为 0 时，分页功能将被禁用（仅对 **Android** 生效）。



```
Future<List<AppInfo>> searchMiniApp(String keyword,
```



```
{int pageIndex = 0, int pageSize = 0})
```

示例代码：



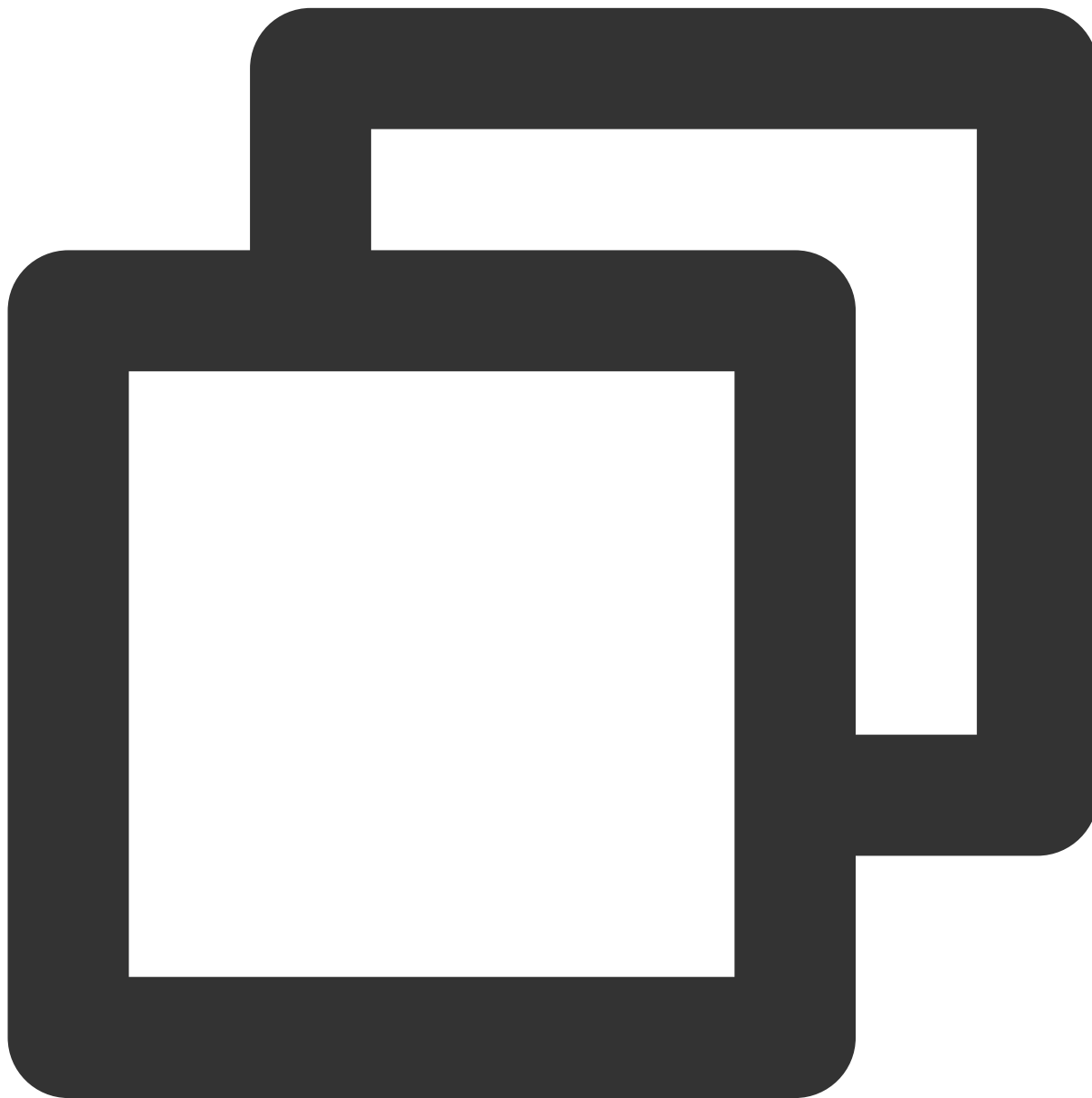
```
Future<void> doSearch() async {  
  List<AppInfo>? infoList;  
  FocusScope.of(context).unfocus();  
  infoList = await _tcmppFlutterPlugin  
    .searchMiniApp(searchController.text)  
    .catchError((err) {  
      ScaffoldMessenger.of(context).showSnackBar(SnackBar(content: Text(err.toString())));  
    });  
}
```

```
setState(() {  
  _appInfoList.clear();  
  if (infoList != null && infoList.isNotEmpty) {  
    _appInfoList.addAll(infoList);  
  } else {  
    _emptyText = 'No mini app found';  
  }  
});  
}
```

获取小程序最近访问列表

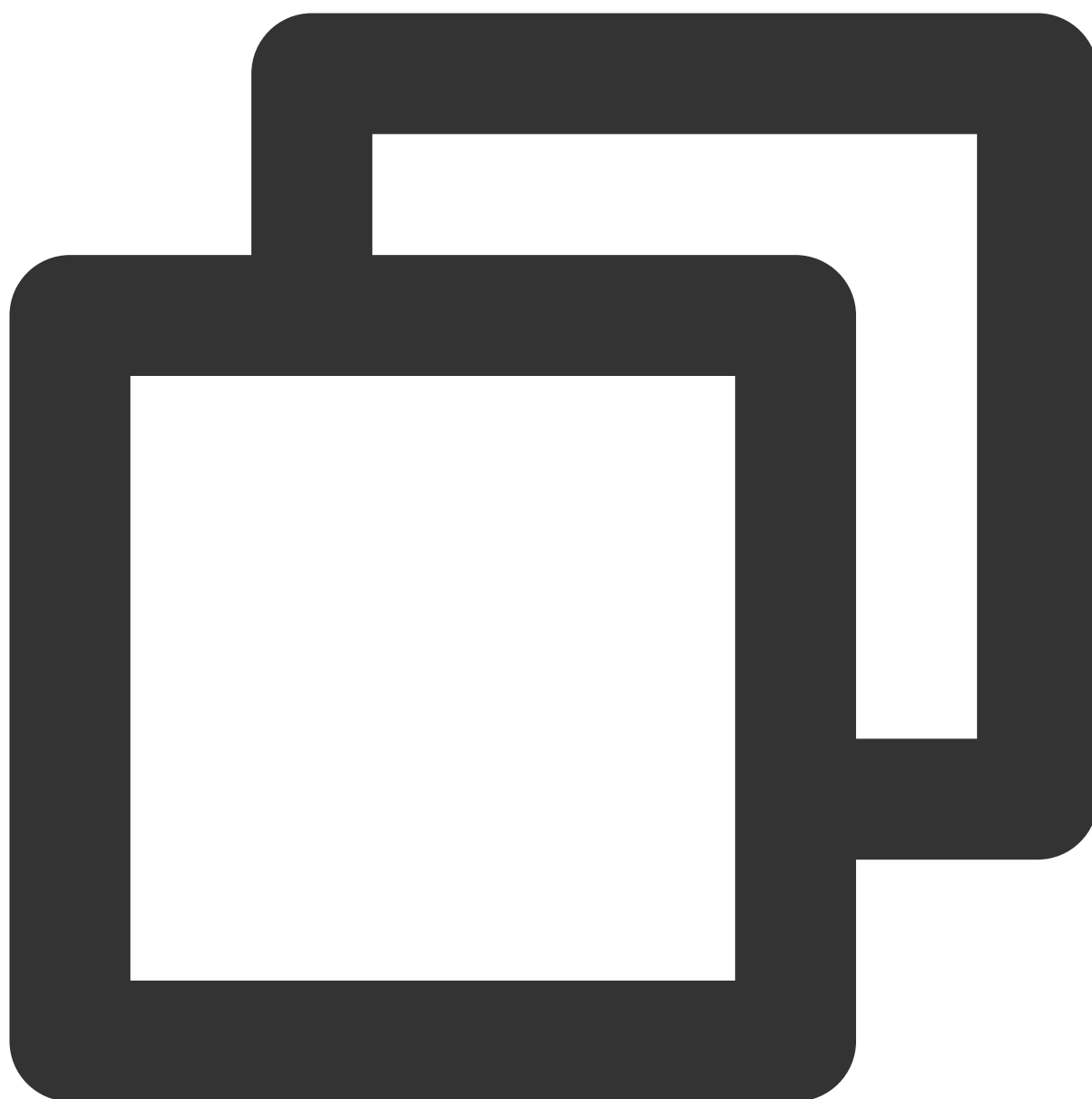
最近更新时间：2024-08-06 15:54:10

tcmpp_flutter插件 提供以下 API 用于访问最近使用的小程序列表。



```
Future<List<AppInfo>?> getRecentList()
```

示例代码：



```
_tcmppFlutterPlugin.getRecentList();
```

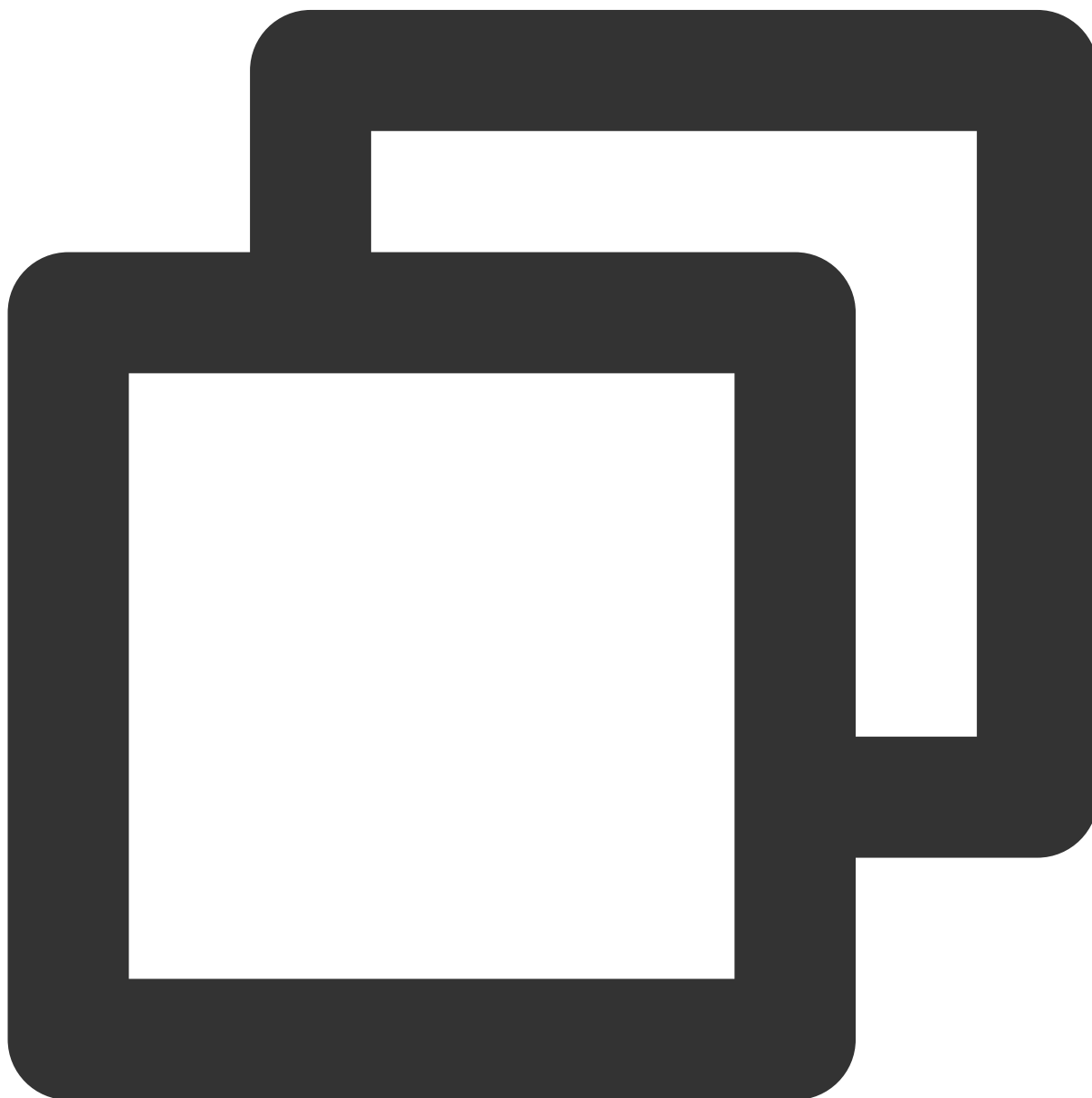
预下载小程序

最近更新时间：2024-08-06 15:51:24

由于小程序在打开时会同步检测和更新版本，小程序在打开过程中可能存在等待的过程；为了降低小程序打开的等待时间，优化用户体验，提供了如下的接口用于预下载小程序包。

appIdList：要加载的小程序 ID 列表。

isDownload：如果设置为“true”，预加载后将下载小程序包，默认为“false”。



```
Future<void> preDownloadMiniApp(List<String> appIdList,
```

```
{bool isDownload = false})
```

示例代码：



```
List<String> appIdList = ['app_id_1', 'app_id_2', 'app_id_3'];  
_tcmppFlutterPlugin.preDownloadMiniApp(appIdList, isDownload: true);
```

自定义小程序能力

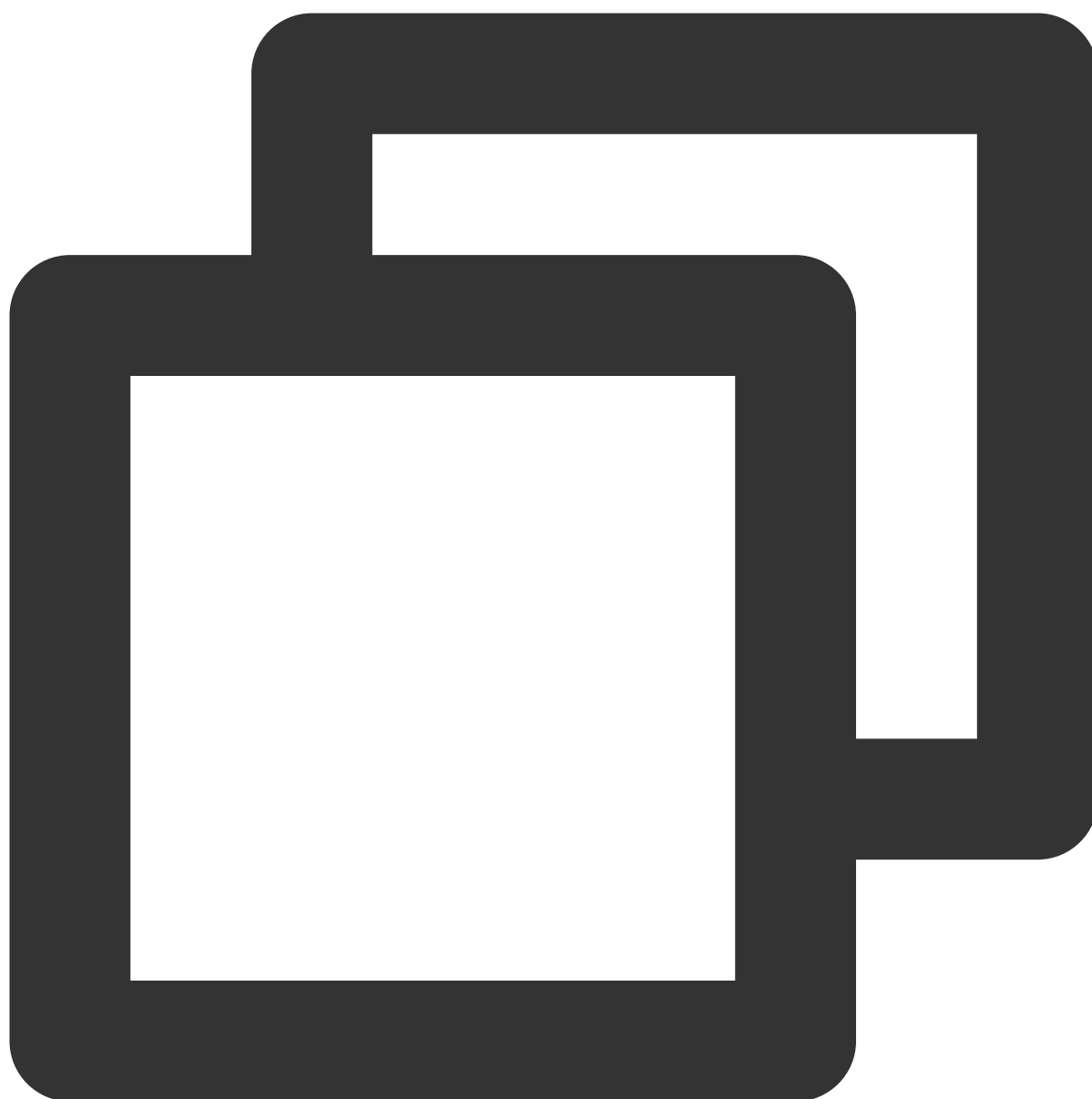
自定义小程序 API

最近更新时间：2024-08-06 15:49:54

如果在小程序中需要调用某些宿主应用提供的能力，在小程序 SDK 未实现或无法实现时，开发者可以通过注册扩展（自定义）API 来实现相关能力，使得小程序能够调用宿主应用中提供的自定义 API 能力。

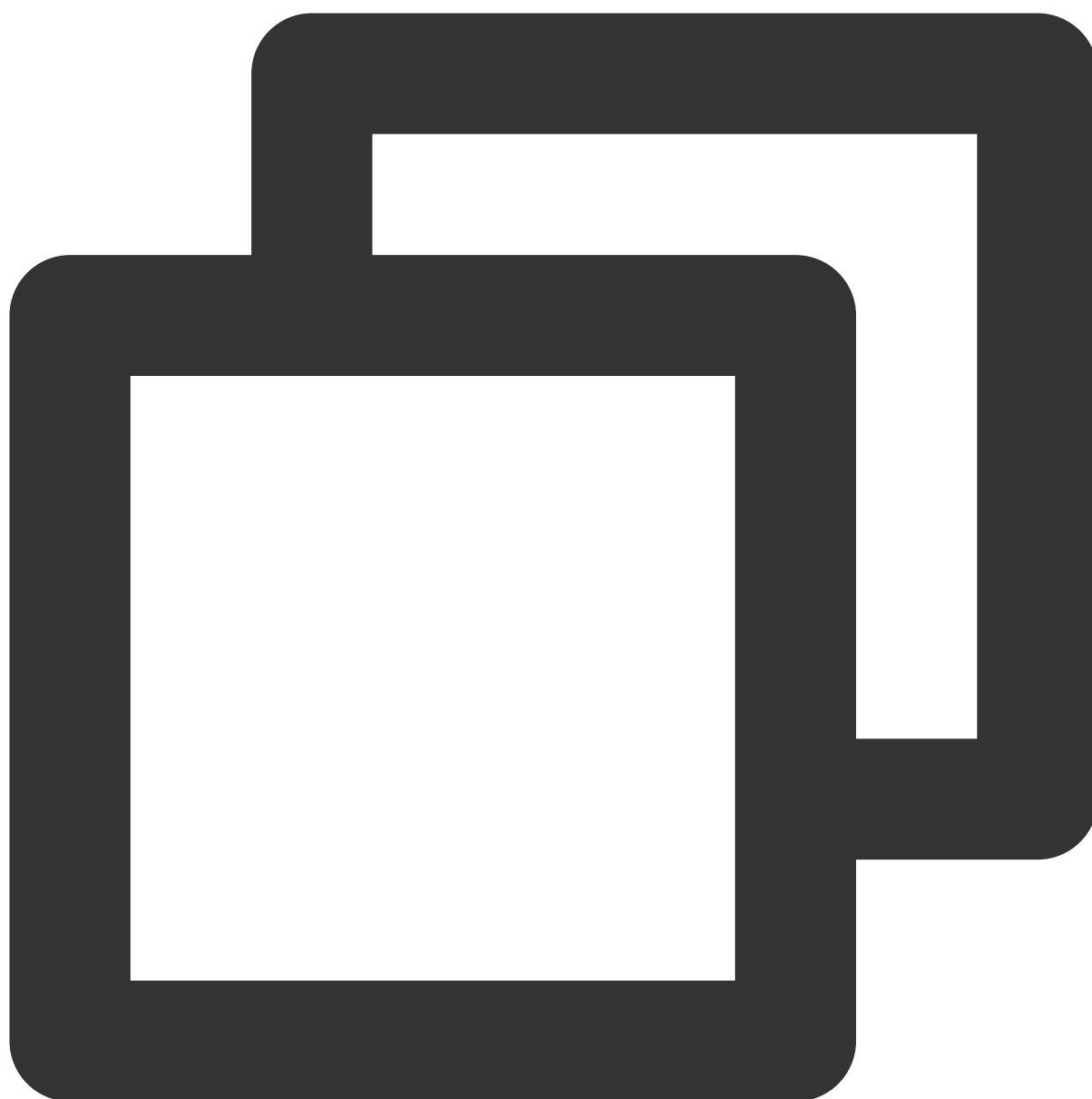
定义自定义的小程序 api 并关联 api 处理程序。**apiName**：自定义 api 的名称，小程序可通过调用 `wx.(apiName)` 使用此 api。

apiHandler: 处理此 api 调用的函数。



```
void registerMiniAppApi(String apiName, TcmppMiniAppApiHandler apiHandler)
```

示例代码：

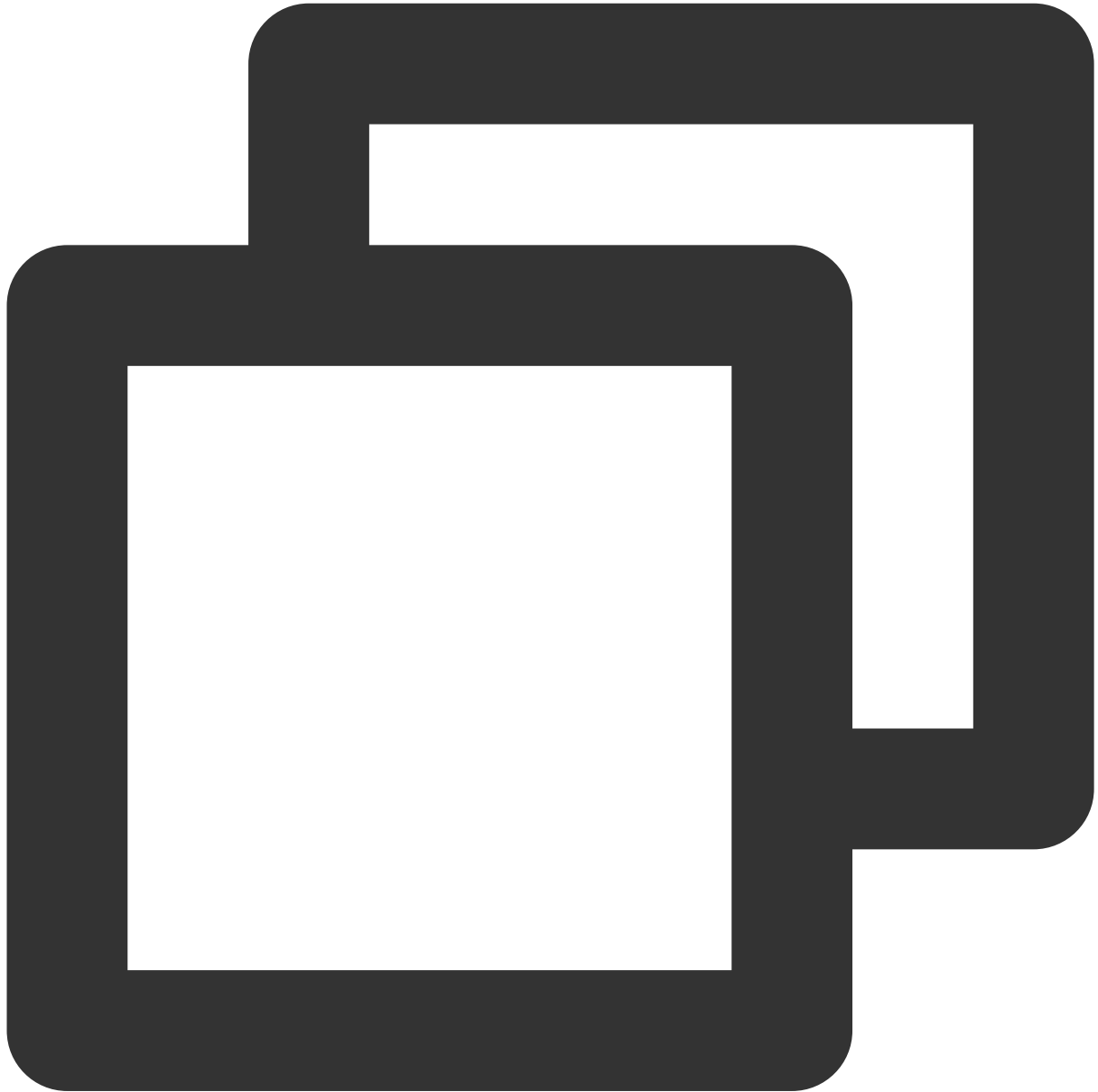


```
final _tcmppFlutterPlugin = TcmppFlutter();  
  
...  
  
@override  
void initState() {  
    super.initState();  
  
    ...  
  
    /// 必须在小程序启动前注册
```

```
_tcmppFlutterPlugin.registerMiniAppApi("myApiName", myApiHandler);  
}
```

小程序会通过 `wx.invokeNativePlugin` 的方式传入 `api_name` 去调用 `flutter` 注册的 `myApiName`。

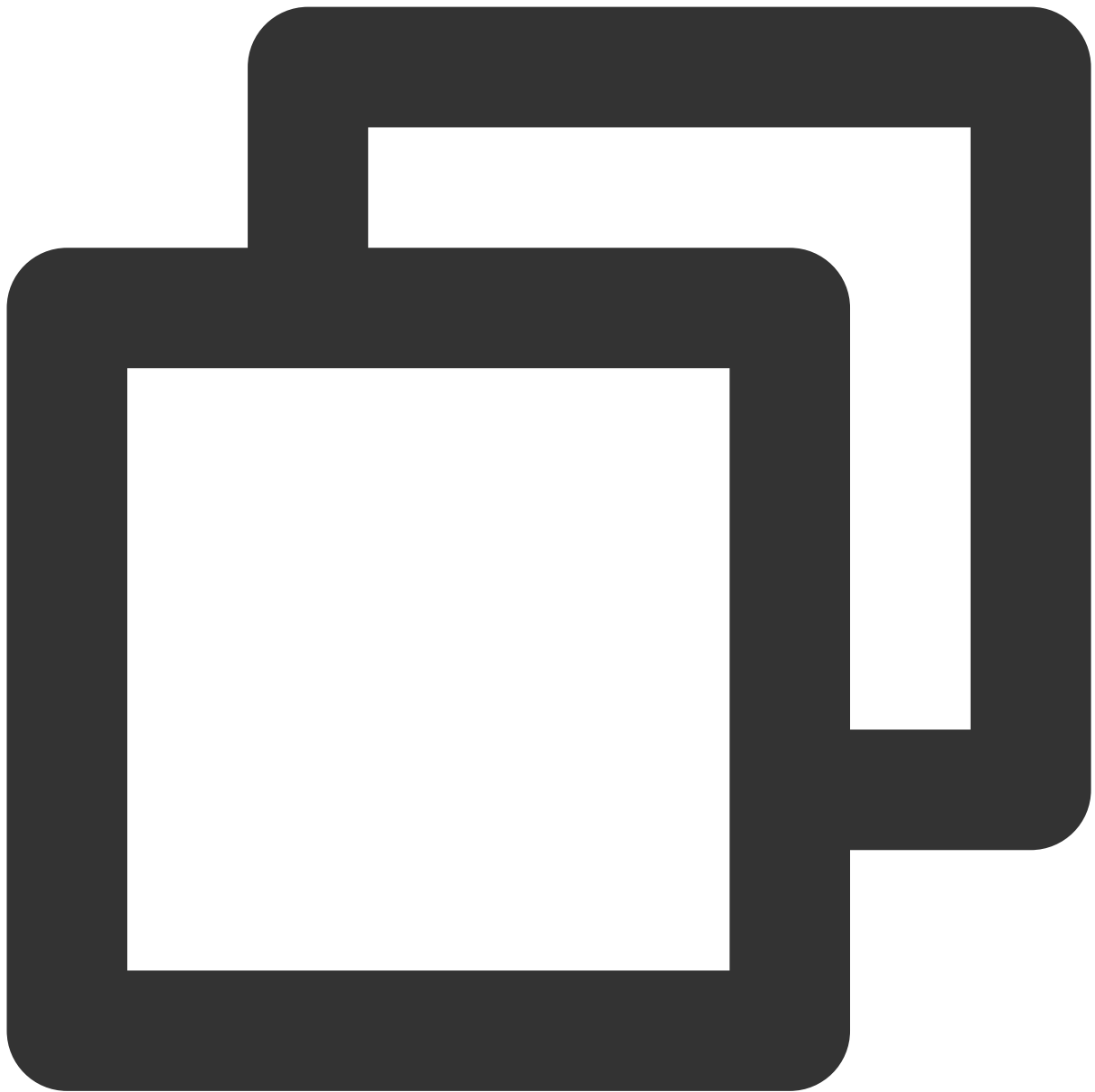
小程序代码的调用示例：



```
example code:  
  /// Mini Program Call  
  var opts = {  
    api_name: 'myApiName',  
    success: function(res) {
```

```
    log(res);
  },
  fail: function(res) {
    log(res);
  },
  complete: function(res) {
    log(res);
  },
  data: {
    name : 'kka',
    age : 22
  }
}
wx.invokeNativePlugin(opts);
```

在 flutter 侧实现 myApiHandler 函数：



```
/// 客户端 API
Future<Map<String, dynamic>?> myApiHandler(MiniApiCall call) async {
  print("API : ${call.apiName}");
  print("AppInfo: ${call.appInfo}");
  print("WebView ID: ${call.webViewId}");
  print("params: ${call.params}");

  return {"result": "success", "method": "myApiHandler"};
}
```

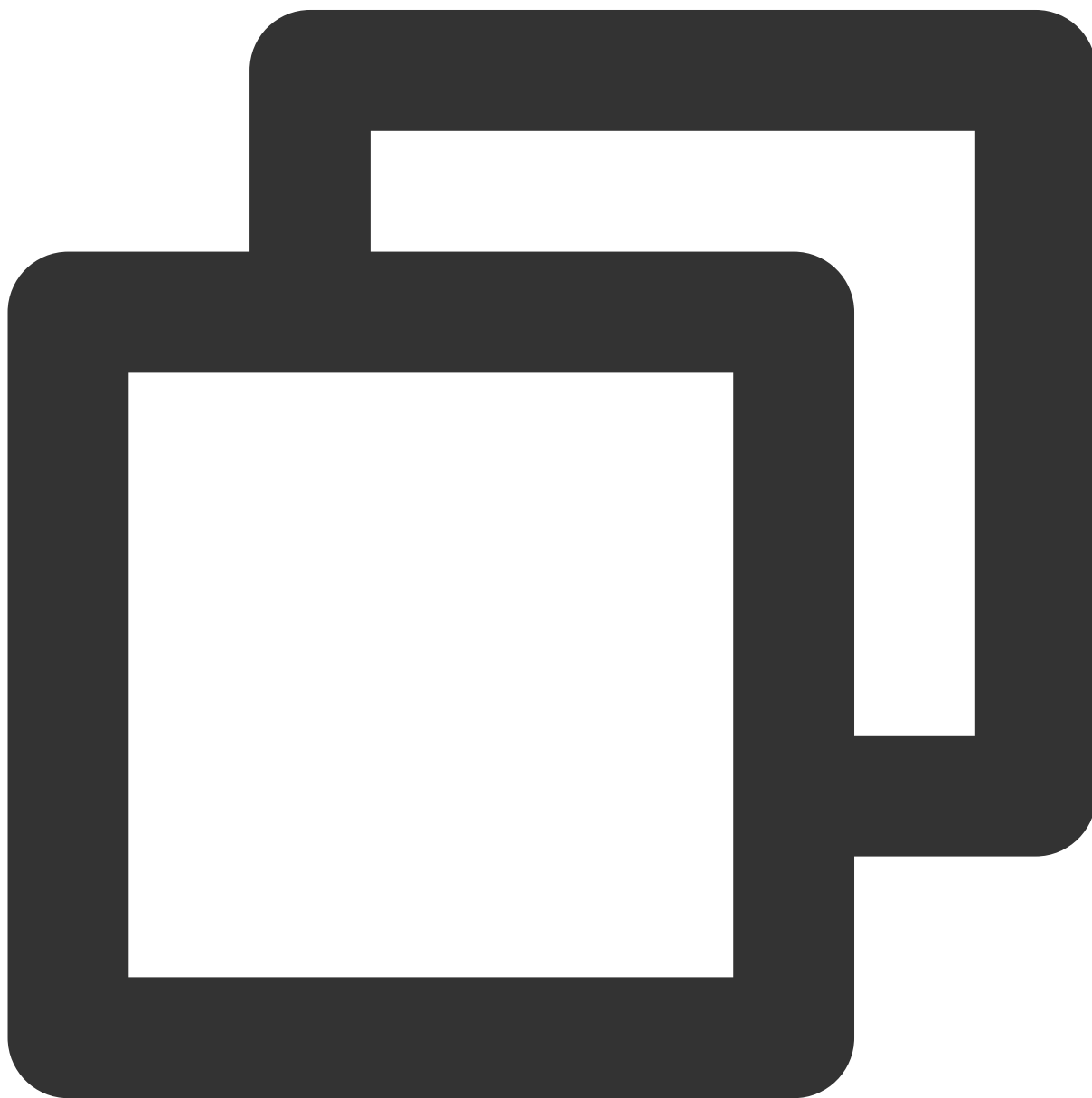

插件定制能力

定制插件能力

最近更新时间：2024-08-06 15:45:09

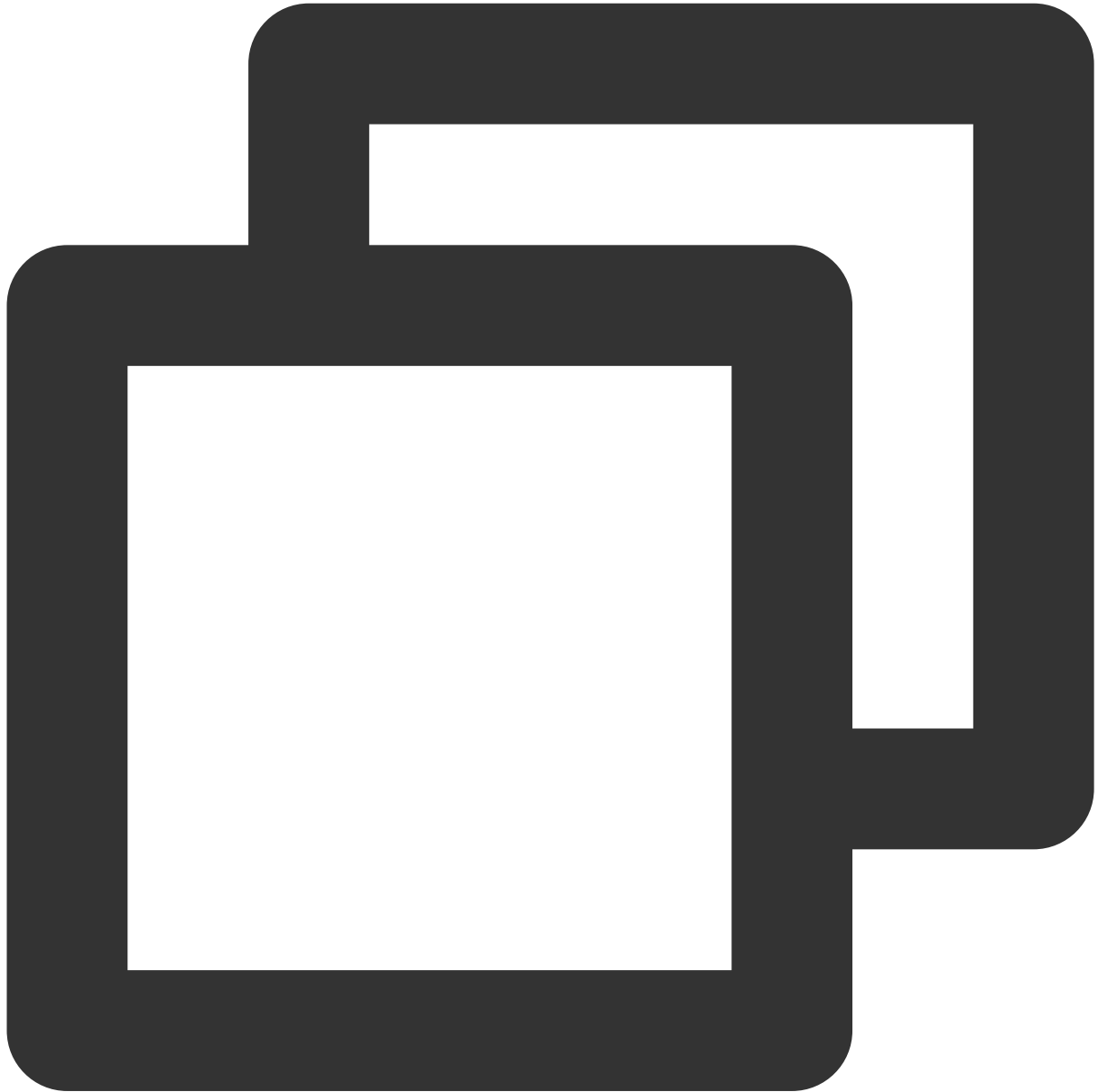
用户可以通过实现 `OpenApiHandler` 和 `TcmppPlatformEventHandler` 抽象类，将小程序和宿主应用的数据交互进行关联。

示例代码：



```
_tcmppFlutterPlugin.registerOpenApiHandler(MyOpenApiHandler());  
_tcmppFlutterPlugin.registerPlatformEventHandler(MyPlatformHandler());
```

平台API类实现示例代码：



```
class MyPlatformHandler extends TcmppPlatformEventHandler {  
  
  @override  
  Future<List<CustomMenu>> getCustomMenus() async {  
    CustomMenu menu1 = CustomMenu(  

```

```
        '100', 'res/images/mini_app_wechat_friend.png', 'Share To', true,
        shareKey: 'twitter');
CustomMenu menu2 = CustomMenu(
    '101',
    'https://staticintl.cloudcachetci.com/cms/backend-cms/8WGP653_%E5%BC%80%E5%'
    'Custom',
    false);

return [
    menu1,
    menu2,
];
}

@Override
Future<void> customMenuClick(String menuId, ShareData? shareMenu) async {
    print("click menuId:$menuId shareMenu:$shareMenu");
    throw UnimplementedError();
}

@Override
Future<bool> reportEvent(int eventId, String eventName, AppInfo appInfo,
    Map<Object?, Object?> params) async {
    print("reportEvent:$eventName appinfo:$appInfo params:$params");
    // TODO: implement reportEvent
    return true;
}

@Override
Future<void> onMiniProgramStateChange(
    String appId, MiniProgramState state) async {
    print("app state change: appid=$appId, state=$state");
}
}
```

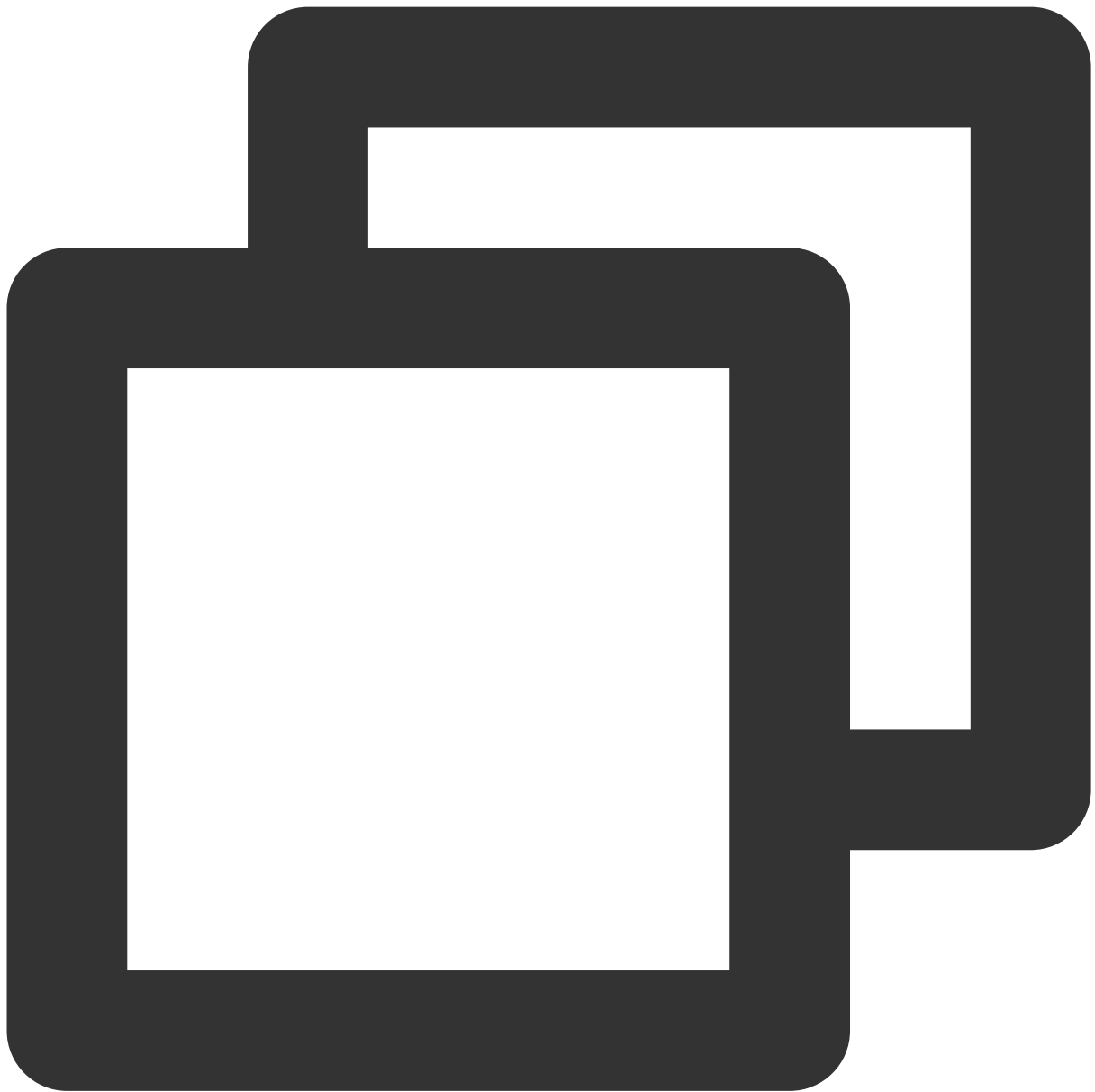

定制小程序信息接口

最近更新时间：2024-08-06 15:35:57

监听小程序生命周期

当小程序启动、进入前台、进入后台和关闭时，Flutter 应用程序主机可通过重载 `onMiniProgramStateChange` 方法获得通知。

示例代码：

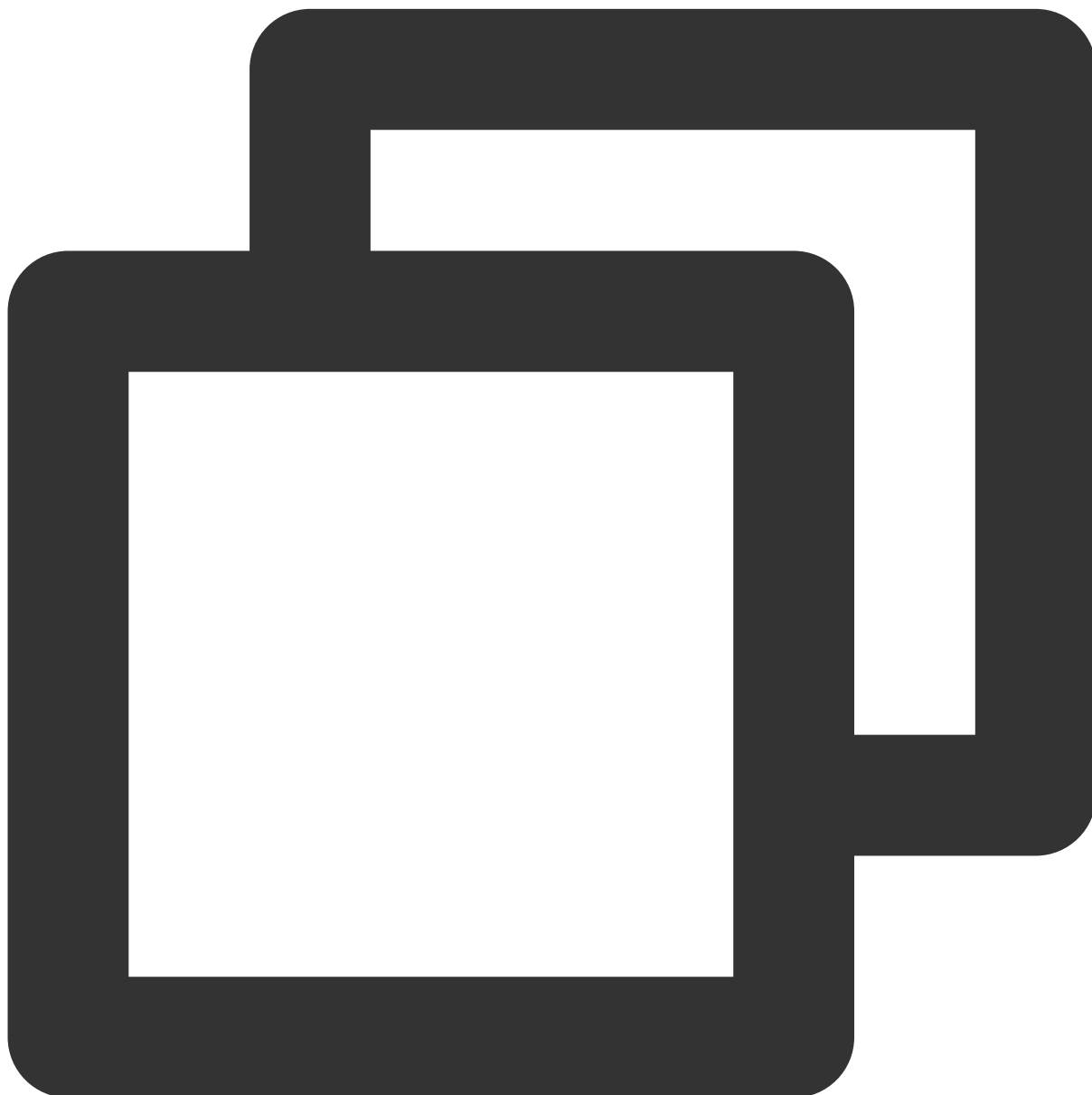


```
@override
Future<void> onMiniProgramStateChange(
    String appId, MiniProgramState state) async {
    print("app state change: appid=$appId, state=$state");
}
```

定制分享能力

最近更新时间：2024-08-06 15:35:11

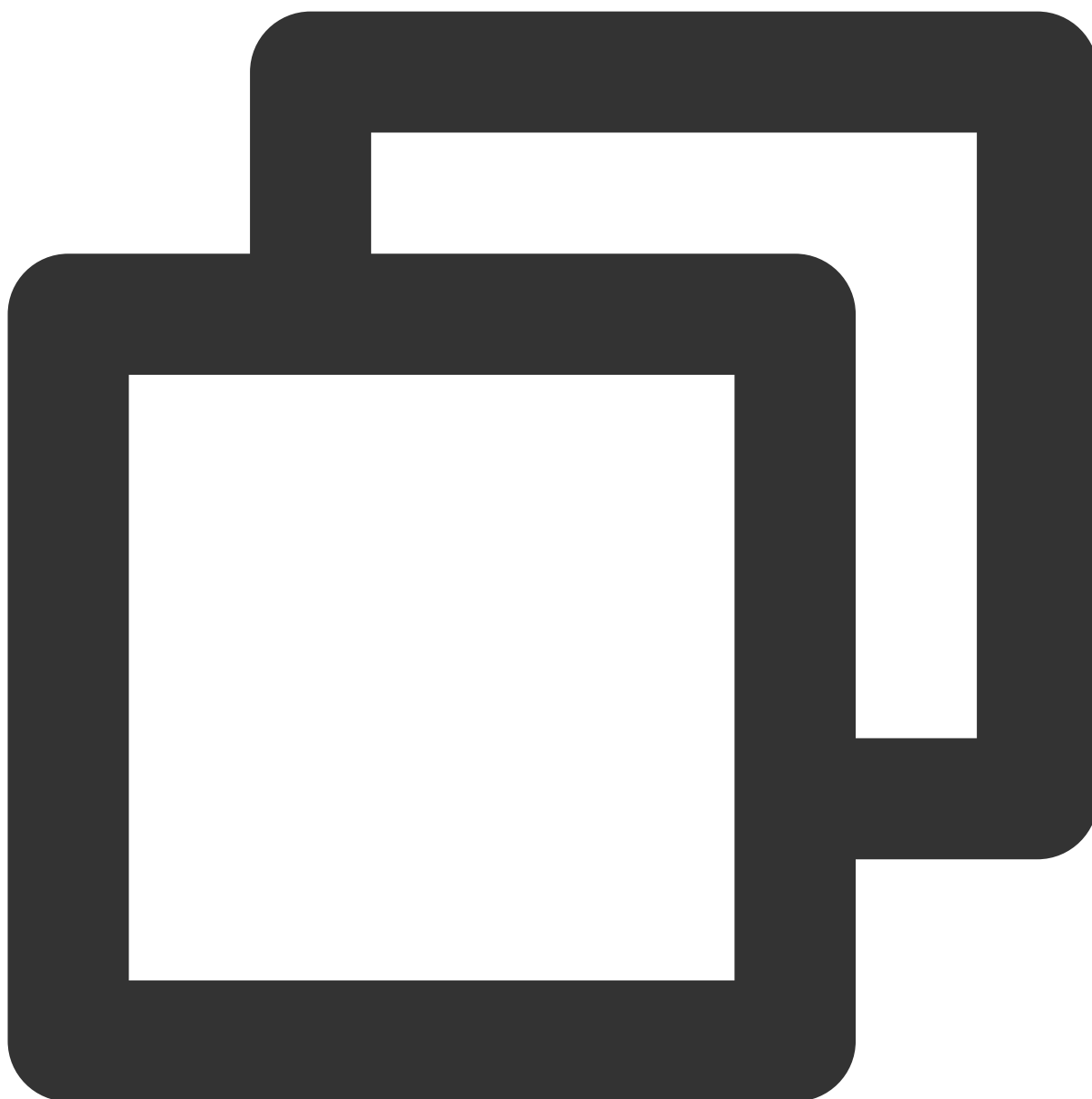
开发人员可以自定义小程序操作面板中的按钮。其中包括分享按钮和其他功能按钮。目前，点击小程序右上方的更多按钮即可调出操作面板：



```
@override
Future<List<CustomMenu>> getCustomMenus() async {
  /// 菜单包含menuId、图片（支持本地路径和网络图片）、菜单名称、是否为自定义分享类型。
```

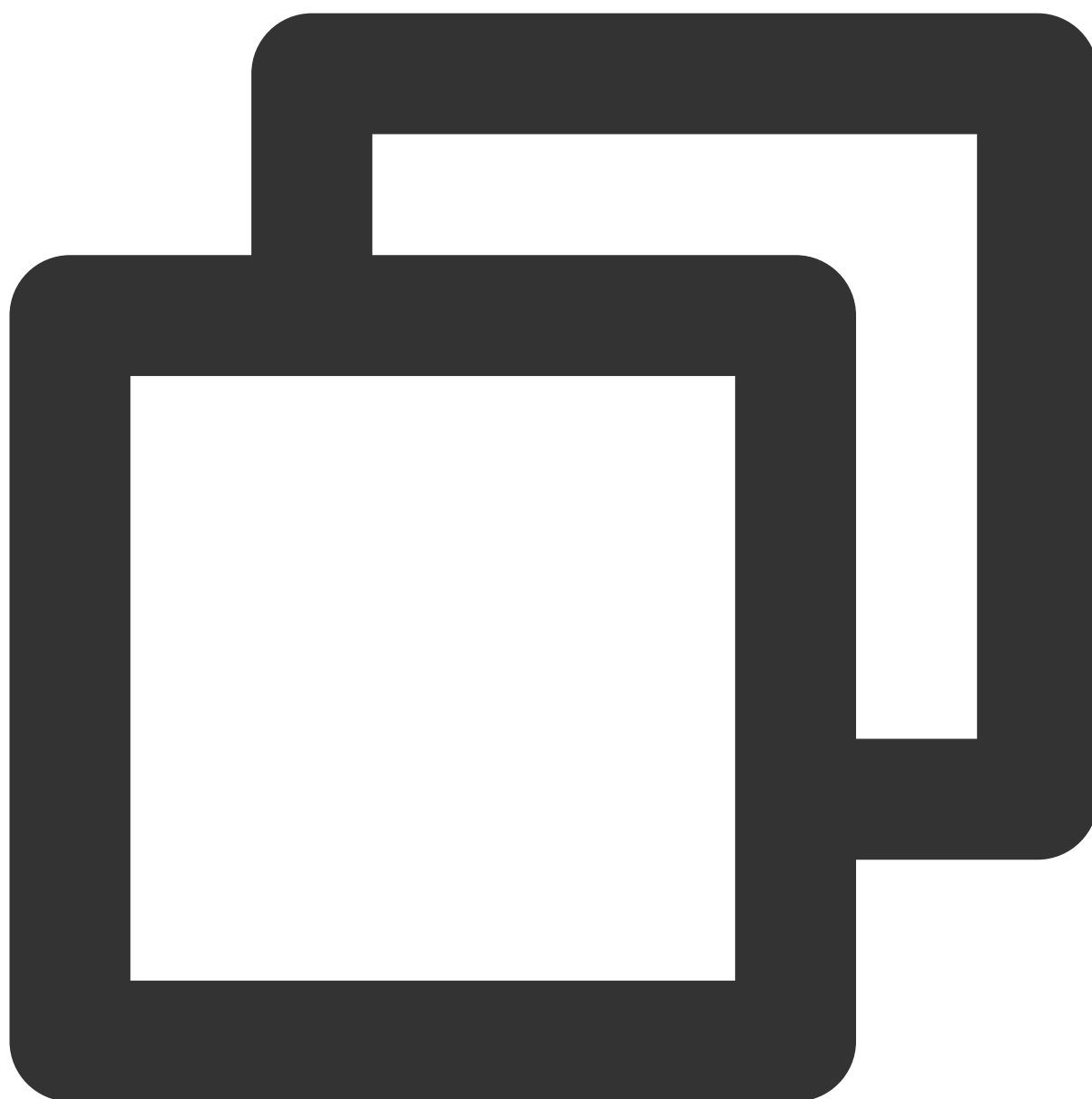
```
CustomMenu menu1 = CustomMenu(  
    '100', 'res/images/mini_app_wechat_friend.png', 'Share To', true,  
    shareKey: 'twitter');  
CustomMenu menu2 = CustomMenu(  
    '101', 'https://staticintl.cloudcachetci.com/cms/backend-cms/8WGP653_%E5%BC%8  
  
return [  
    menu1,  
    menu2,  
];  
}
```

点击菜单回调：



```
@override
Future<void> customMenuClick(String menuId, ShareData? shareMenu) async {
  /// 点击菜单按钮的回调
  print("click menuId:$menuId shareMenu:$shareMenu");
  throw UnimplementedError();
}
```

菜单内内置了QQ，QQ空间，微信，微信朋友圈四个分享渠道的按钮，如需展示这些分享按钮，可以通过以下方法进行配置。



```
_tcmppFlutterPlugin.defaultShareTargets([ShareTarget.qq, ShareTarget.wxMoment]);
```

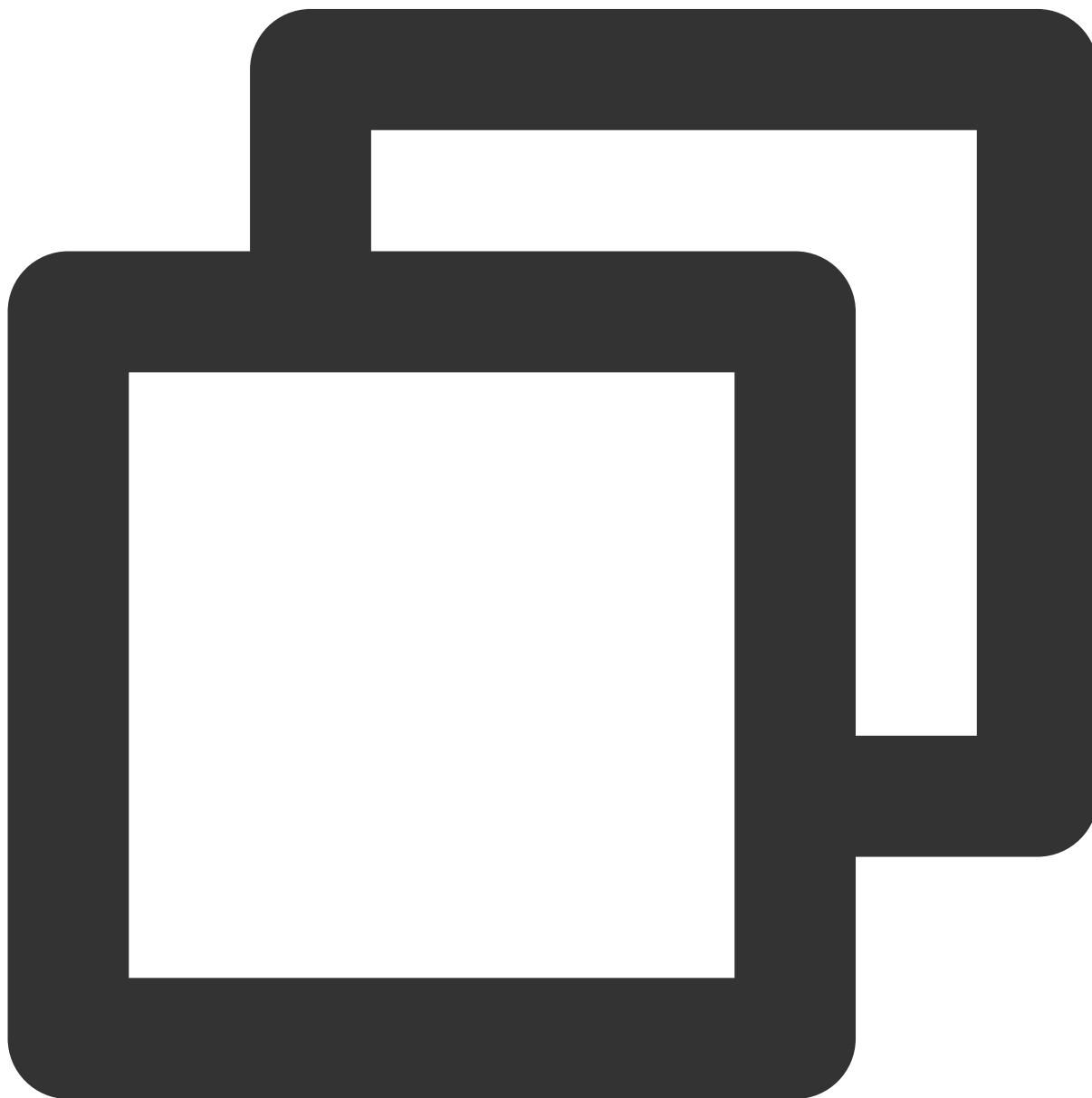
定制用户属性

最近更新时间：2024-08-06 15:31:35

设置当前登录用户的账号信息，用于数据隔离和用户信息展示。

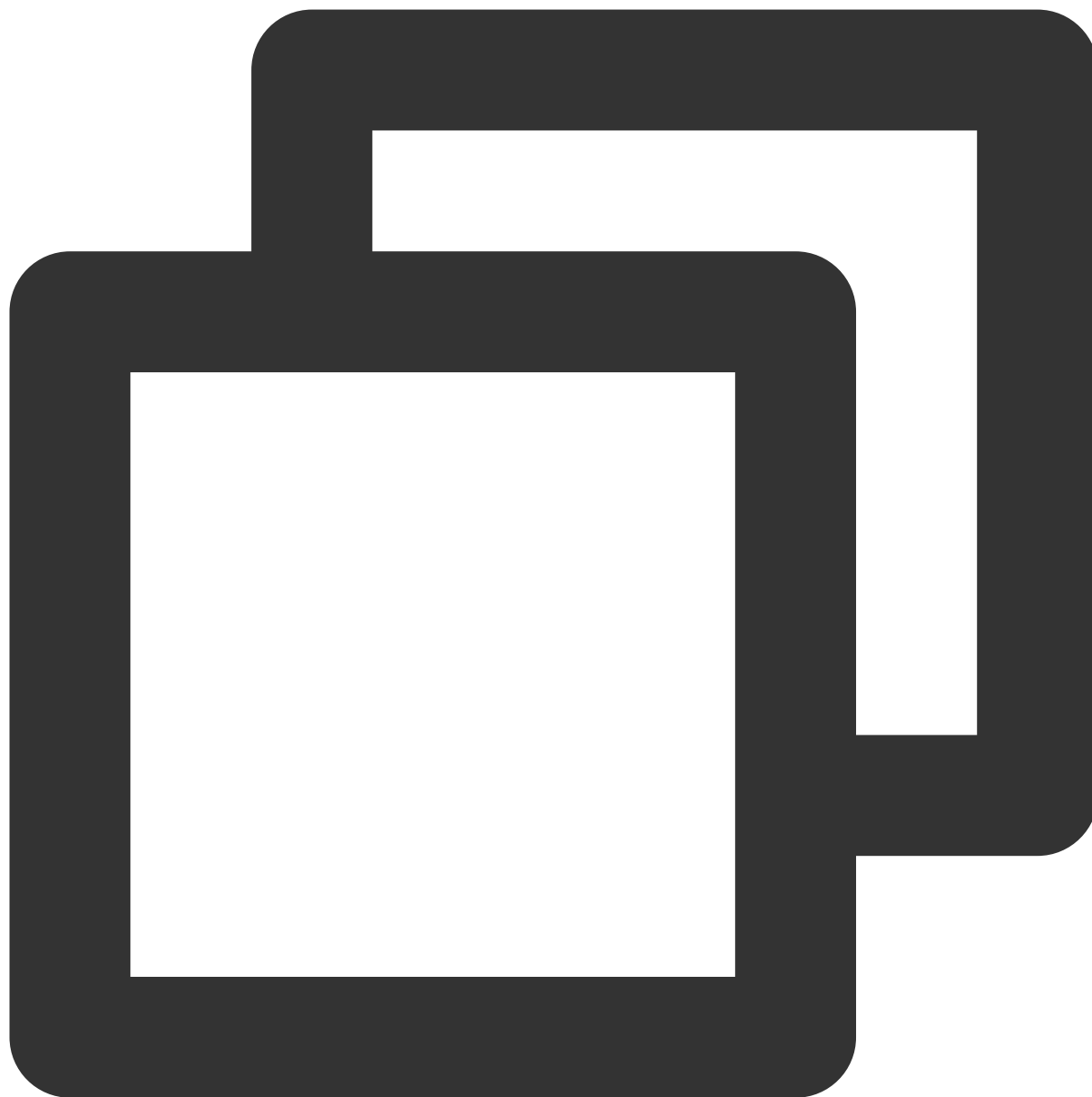
小程序启动前必须设置：

info：当前登录账户的信息，未登录则为空。



```
void setAccount (AccountInfo? info)
```

示例代码：

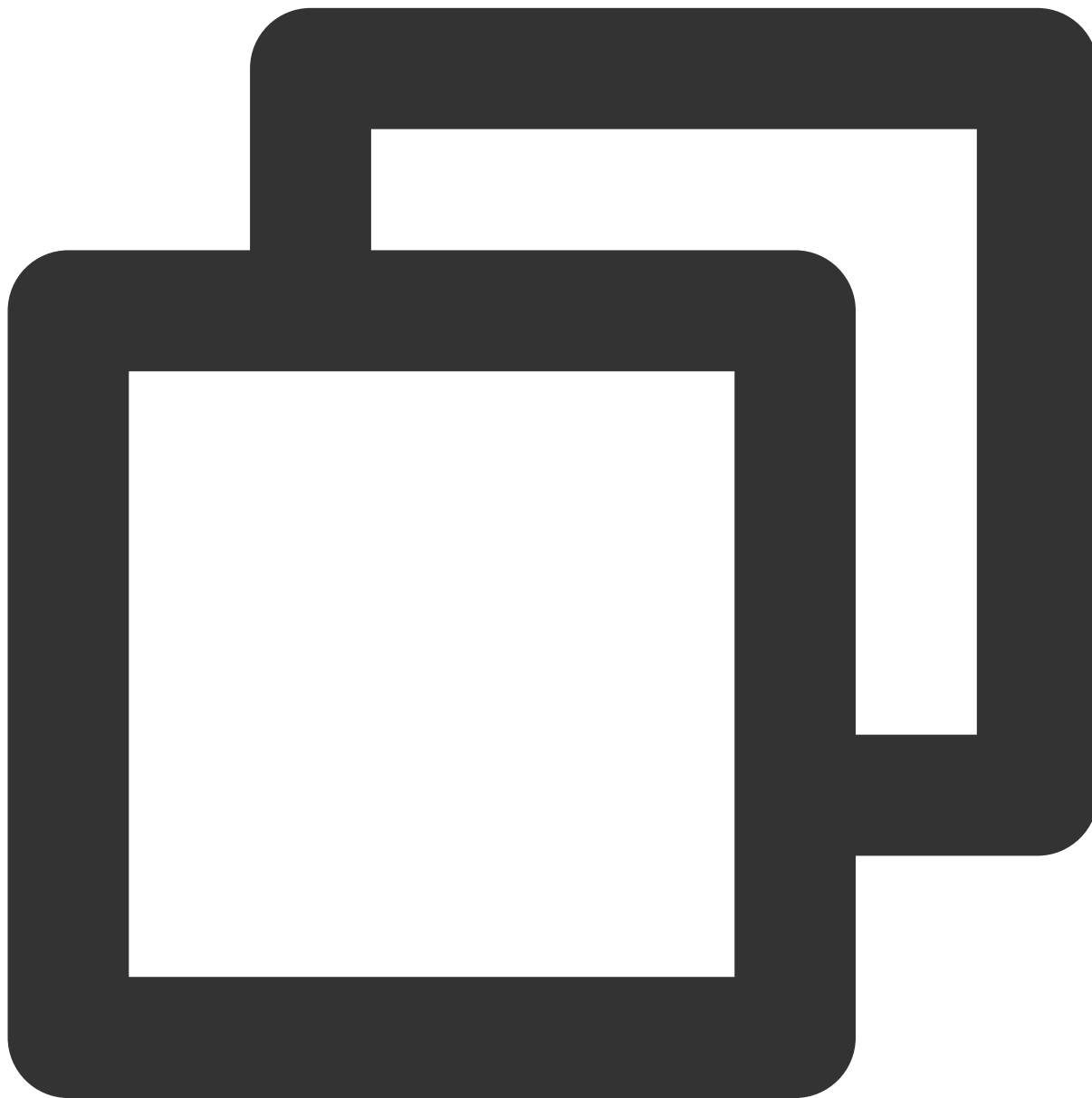


```
_tcmppFlutterPlugin.setAccount (AccountInfo (  
  uid: "12345",  
  avatarUrl: "https://picsum.photos/250?image=9",  
  accountName: "SimpleAccount"));
```


事件上报

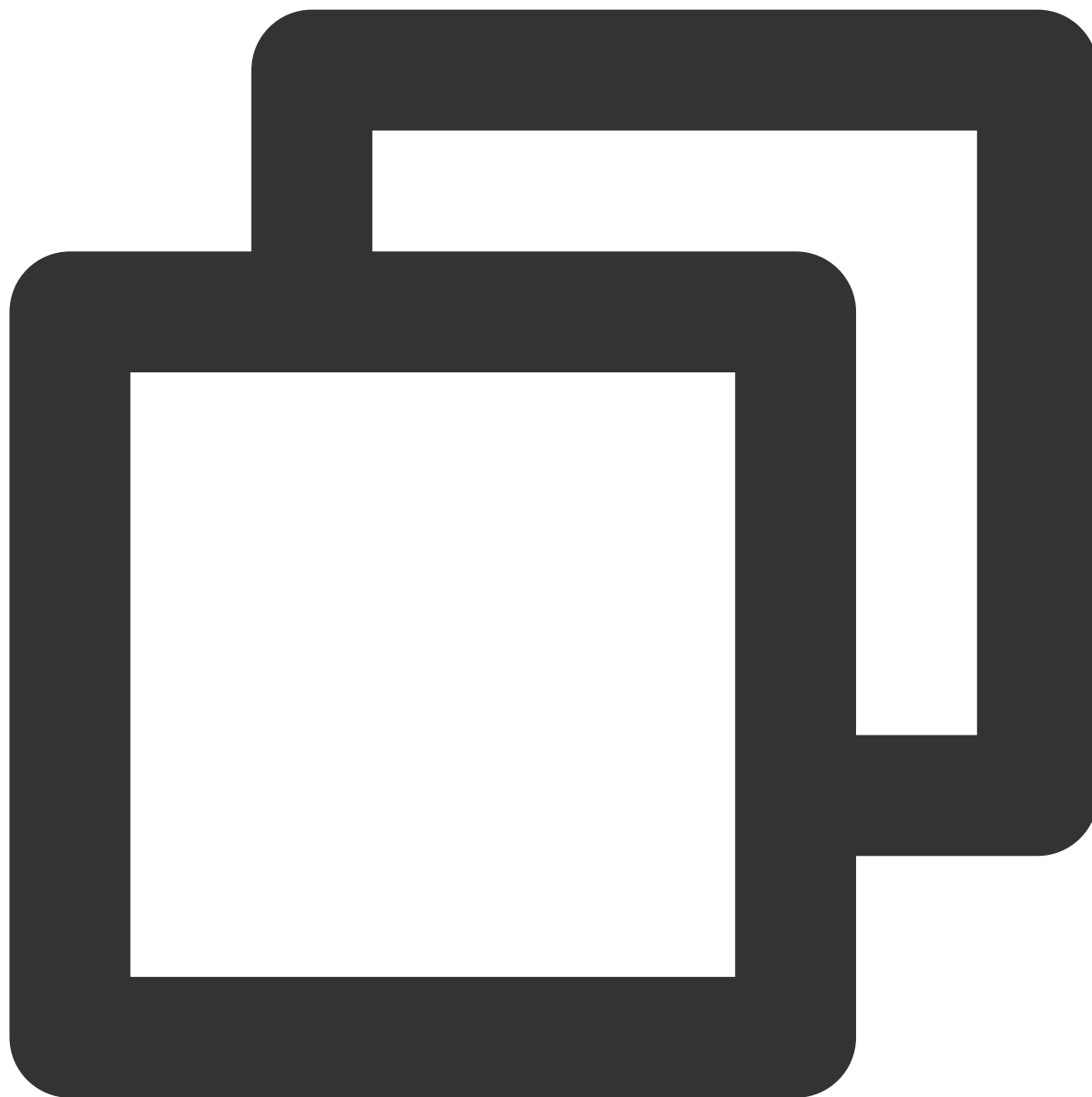
最近更新时间：2024-08-06 15:24:54

宿主 App 可以实现事件上报接口，来覆盖 tcmpp_flutter 内部的上报逻辑。



```
Future<bool> reportEvent(int eventId, String eventName, AppInfo appInfo,  
    Map<Object?, Object?> params) async {  
    return false;  
}
```

示例代码：



```
@Override
Future<bool> reportEvent(int eventId, String eventName, AppInfo appInfo,
    Map<Object?, Object?> params) async {
    print("reportEvent:$eventName appinfo:$appInfo params:$params");
    // TODO: implement reportEvent
    return true;
}
```

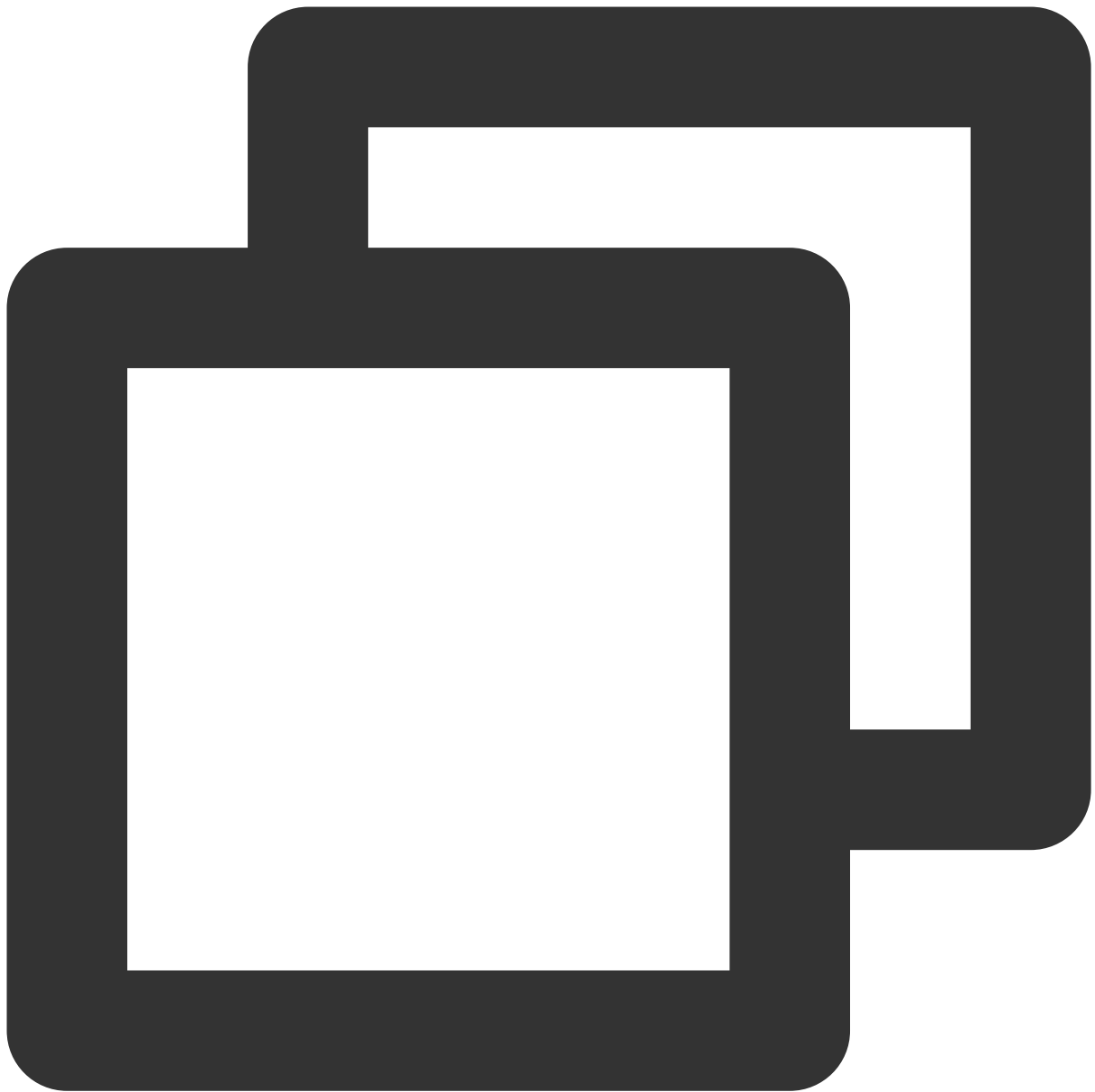
开放接口

最近更新时间：2024-08-06 15:23:27

小程序 SDK 提供部分开放接口，用于调用和实现登录、获取用户信息、支付等宿主应用提供的能力；当前支持的开放接口如下表格所示：

小程序方法	MiniOpenApiProxy 方法	描述
wx.login	login	登陆接口
wx.getUserInfo	getUserInfo	获取用户基本信息
wx.getUserProfile	getUserProfile	获取用户属性信息
wx.getPhoneNumber	getPhoneNumber	获取电话号码
wx.requestPayment	requestPayment	发起支付
wx.checkSession	checkSession	检查登陆状态是否过期

用户可以通过实现 OpenApiHandler 抽象类，将小程序和宿主应用的数据交互进行关联，如下函数定义：



```
abstract class OpenApiHandler {  
    //当小程序调用 wx.requestPayment, 请求第三方支付操作时调用  
  
    Future<Map<String, dynamic>> requestPayment(  
        AppInfo appInfo, Map<Object?, Object?> params);  
  
    /// 当小程序调用 wx.getUserProfile 请求主机应用程序的用户信息时调用  
    ///  
    Future<Map<String, dynamic>> getUserProfile(  
        AppInfo appInfo, Map<Object?, Object?> params);  
}
```

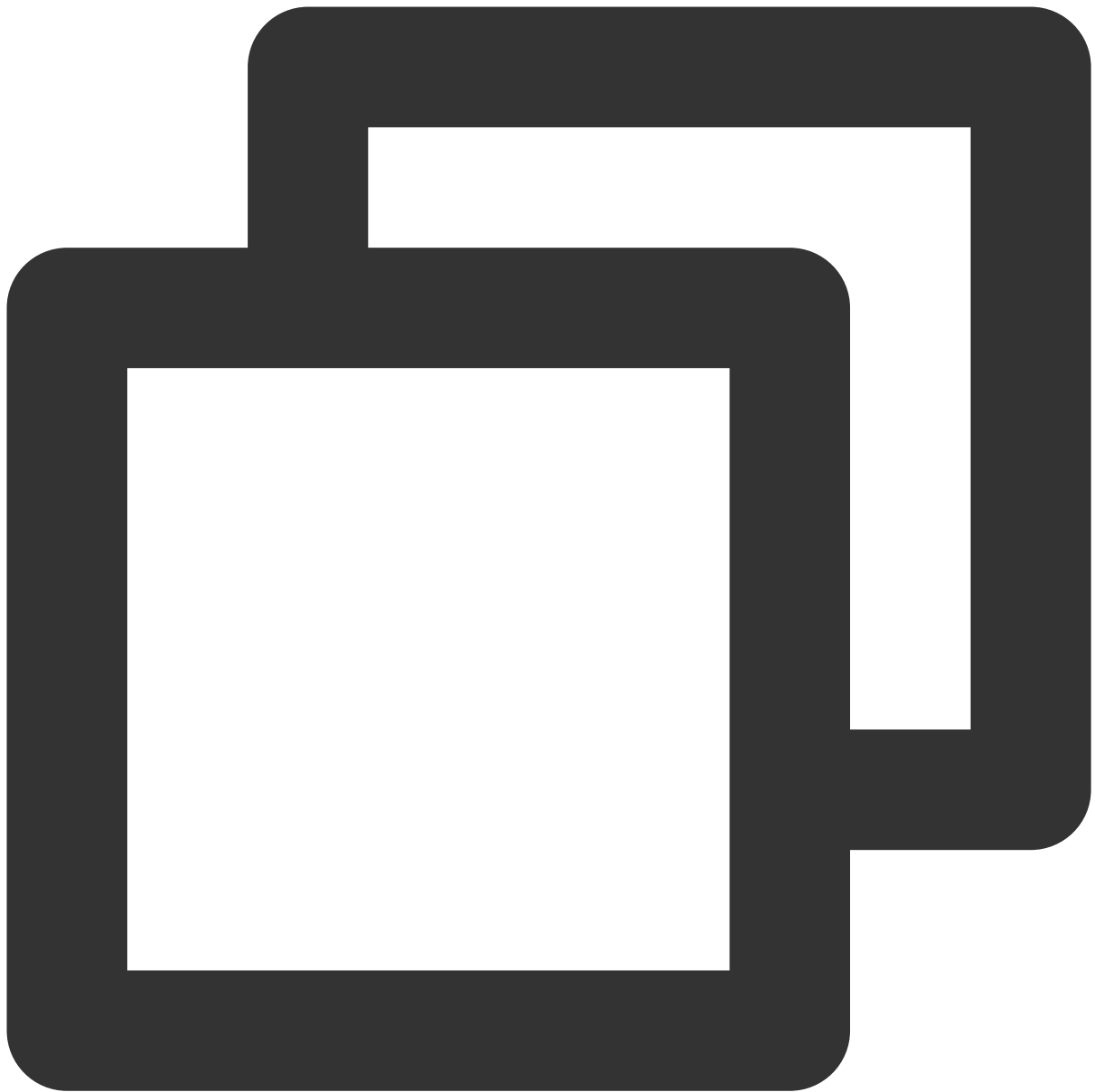
```
/// 当小程序调用 wx.login 时调用，请求主机应用程序的登录证书
///
Future<Map<String, dynamic>> login(
    AppInfo appInfo, Map<Object?, Object?> params);

/// 当小程序调用 wx.checkSession 时调用，请求主机应用程序的登录状态
/// 检查登录是否已过期
///
Future<Map<String, dynamic>> checkSession(
    AppInfo appInfo, Map<Object?, Object?> params);

/// 当小程序调用 wx.getUserInfo 时调用，已被 wx.getUserProfile 过时
/// 兼容早期的小程序 api
///
Future<Map<String, dynamic>> getUserInfo(
    AppInfo appInfo, Map<Object?, Object?> params);

/// 当小程序调用 wx.getPhoneNumber 时调用，获取当前用户的电话号码
///
Future<Map<String, dynamic>> getPhoneNumber(
    AppInfo appInfo, Map<Object?, Object?> params);
}
```

获取用户信息实例代码：



```
@Override
Future<Map<String, dynamic>> getUserProfile(
    AppInfo appInfo, Map<Object?, Object?> params) async {
    print("getUserProfile:$appInfo params:$params");
    Map<String, dynamic> result = {
        "userInfo": {
            "nickName": "xcode",
            "avatarUrl":
                "https://staticintl.cloudcachetci.com/cms/backend-cms/8WGP653_%E5%BC%80",
            "gender": 1,
            "country": "China",
```

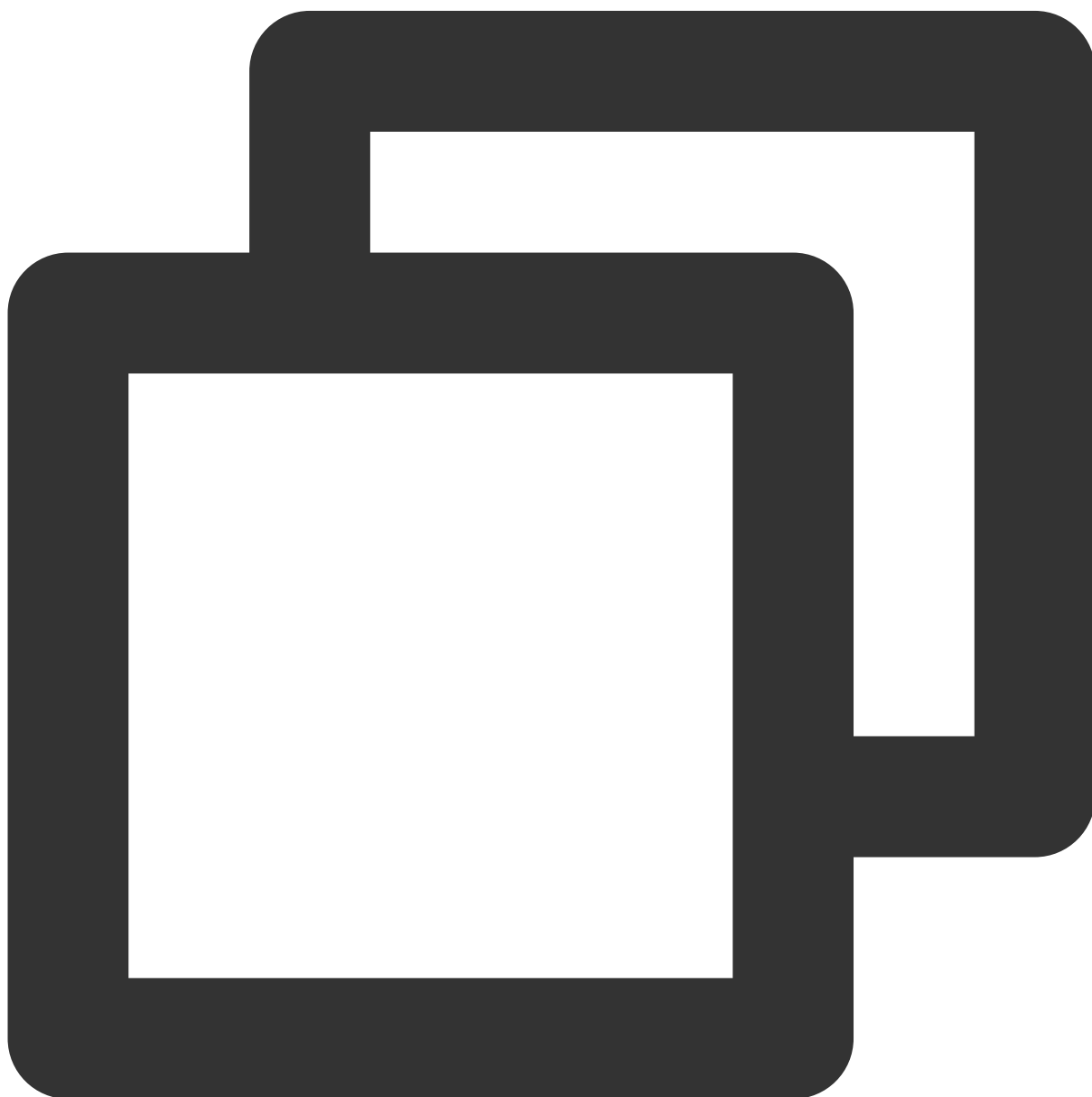
```
        "province": "ChongQing",  
        "city": "ChongQing",  
    }  
};  
return Future.value(result);  
}
```

其它

最近更新时间：2024-08-06 15:23:05

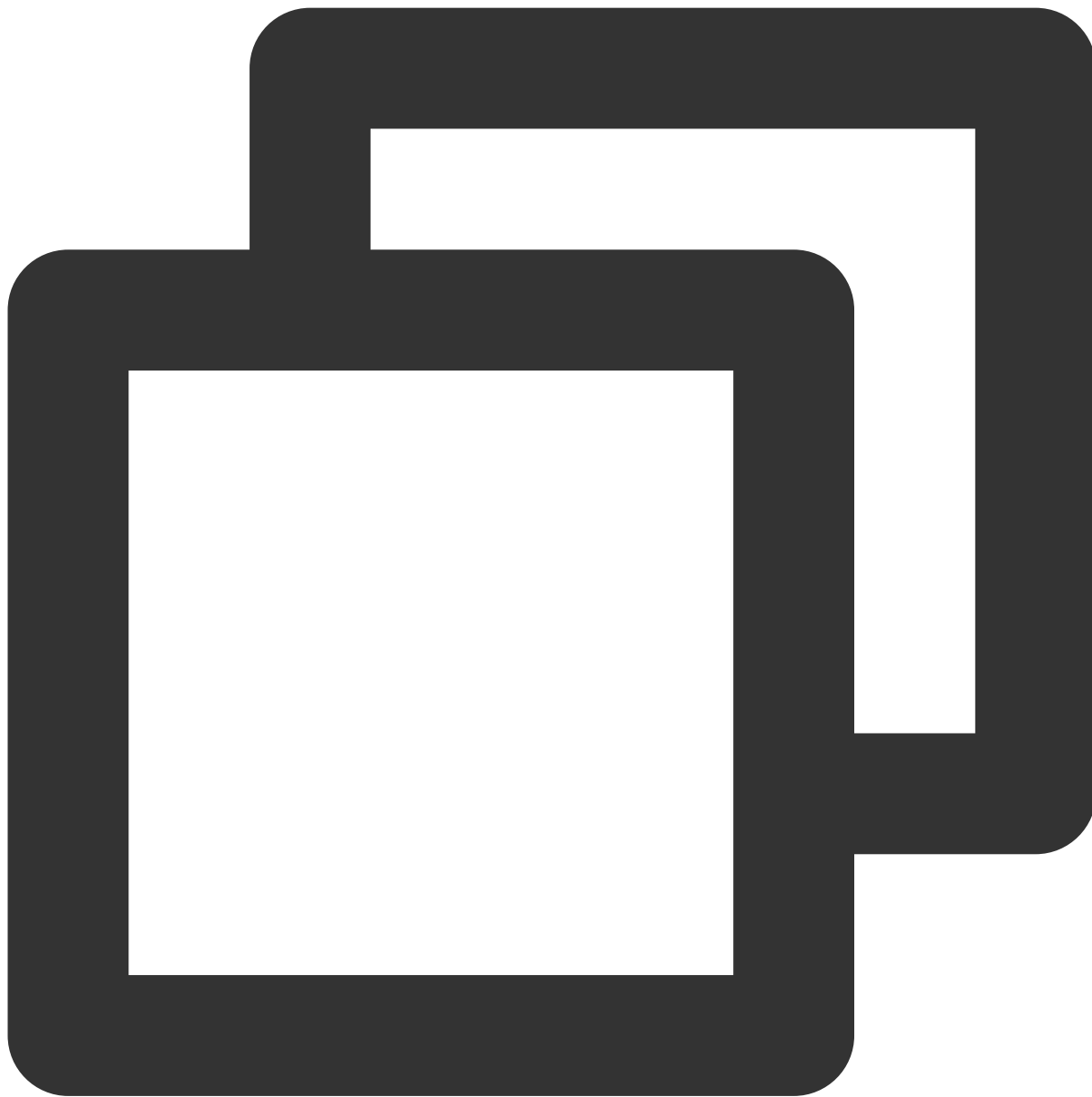
设置小程序容器的主题

必须在小程序启动前设置主题：浅色模式、深色模式或使用系统设置。



```
Future<void> setTheme (MiniTheme theme)
```


示例代码：



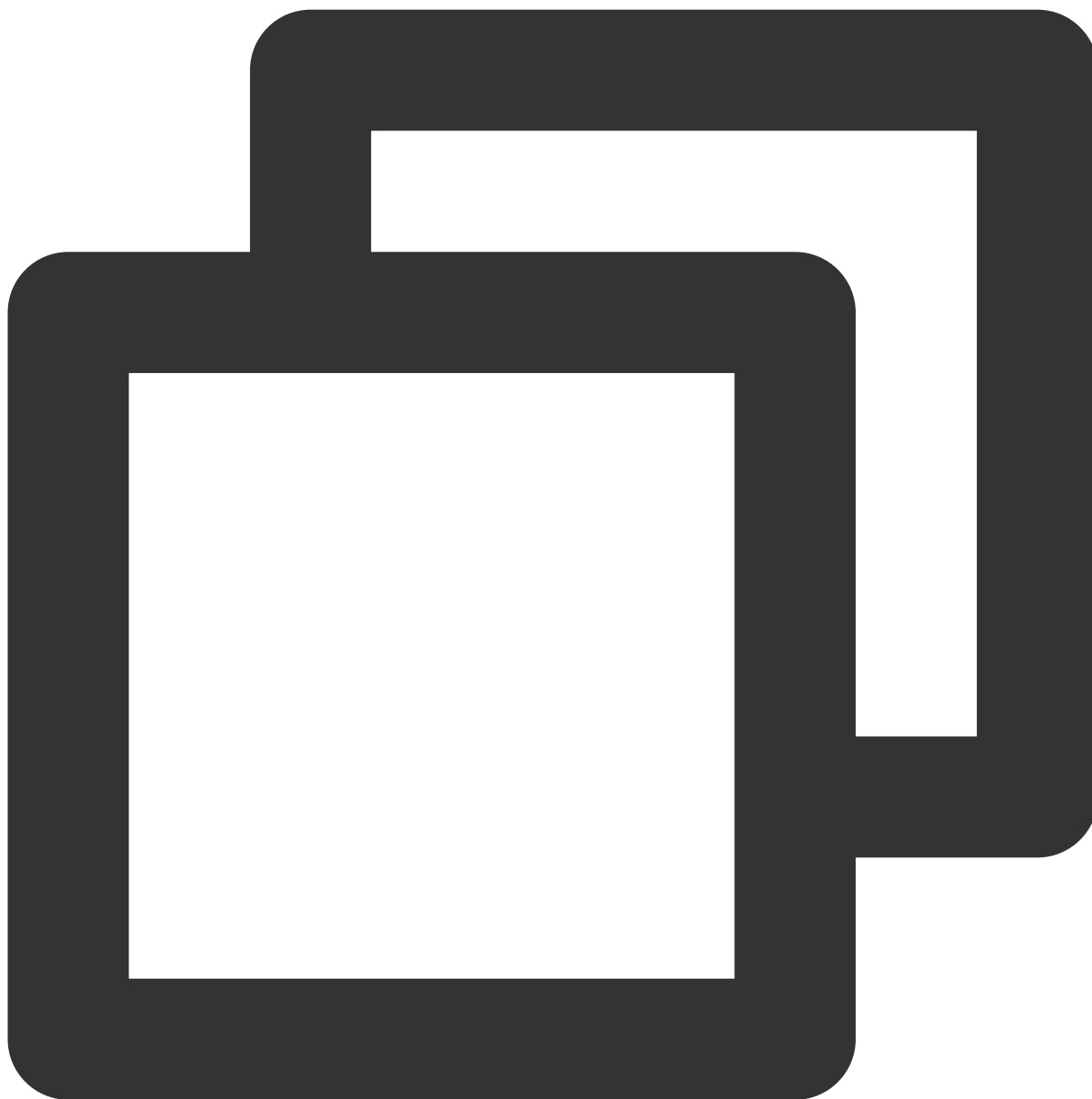
```
_tcmpFlutterPlugin.setTheme(MiniTheme.dark);
```

设置小程序的本地化

必须在小程序启动前设置。

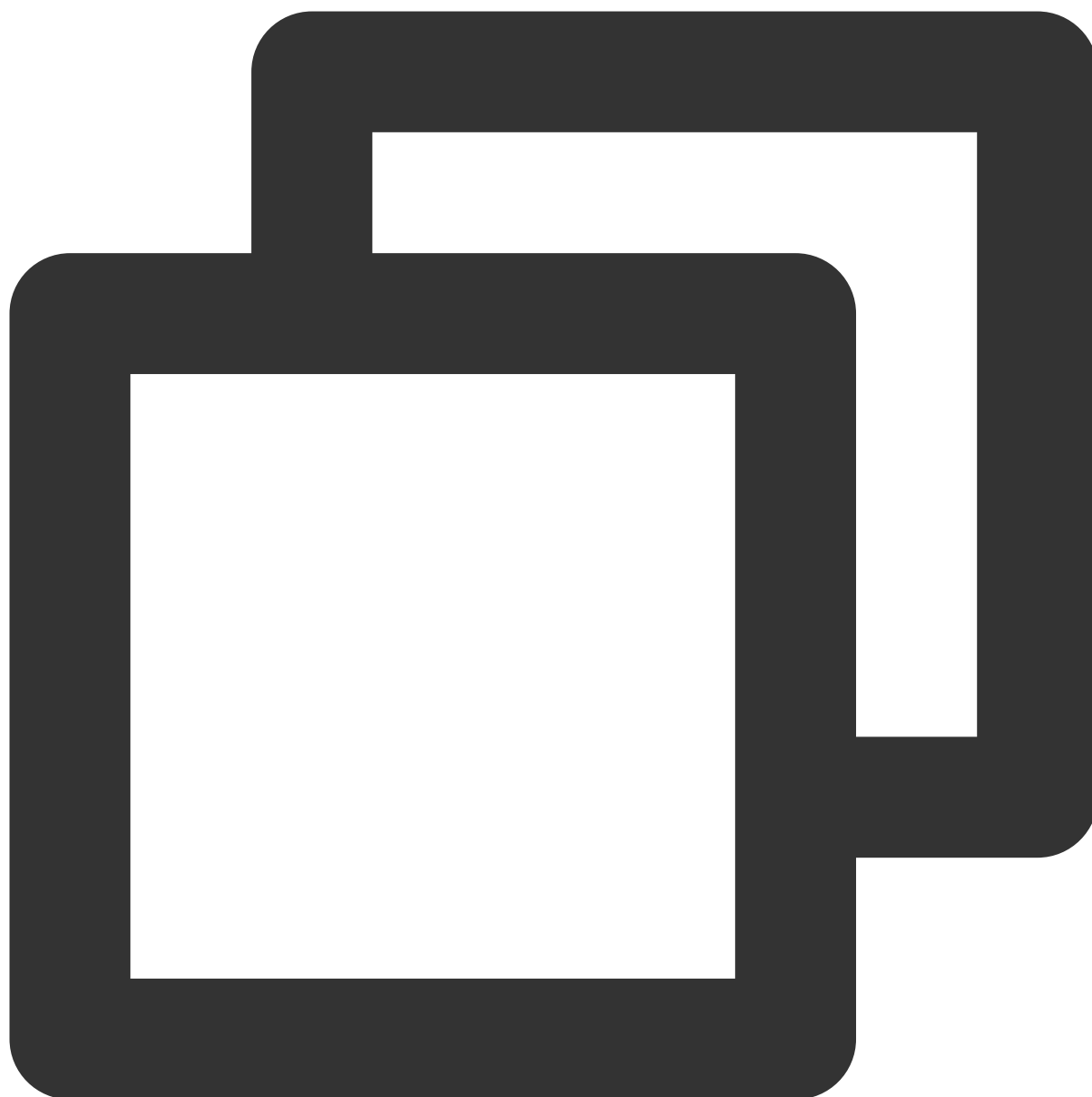
language：小程序的语言。应为 ISO 639 alpha-2 或 alpha-3 语言代码。应为 ISO 3166 alpha-2 国家代码。

variant：任意值，用于表示本地语言的变体。



```
Future<void> setLocale(String language,  
    {String? region, String? variant})
```

示例代码：

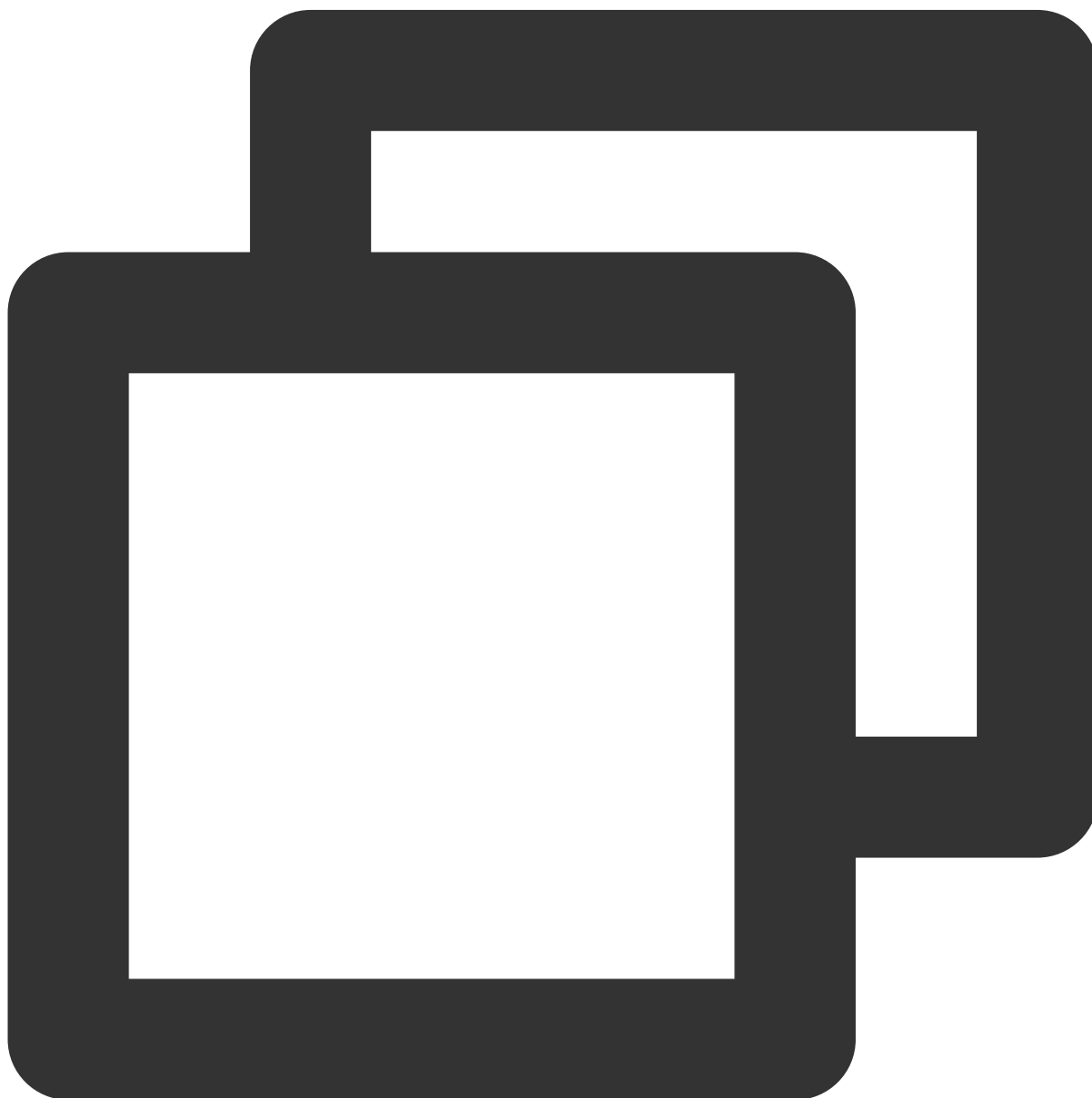


```
_tcmpFlutterPlugin.setLocale("en", region: "us");
```

API 说明

最近更新时间：2024-08-06 15:17:17

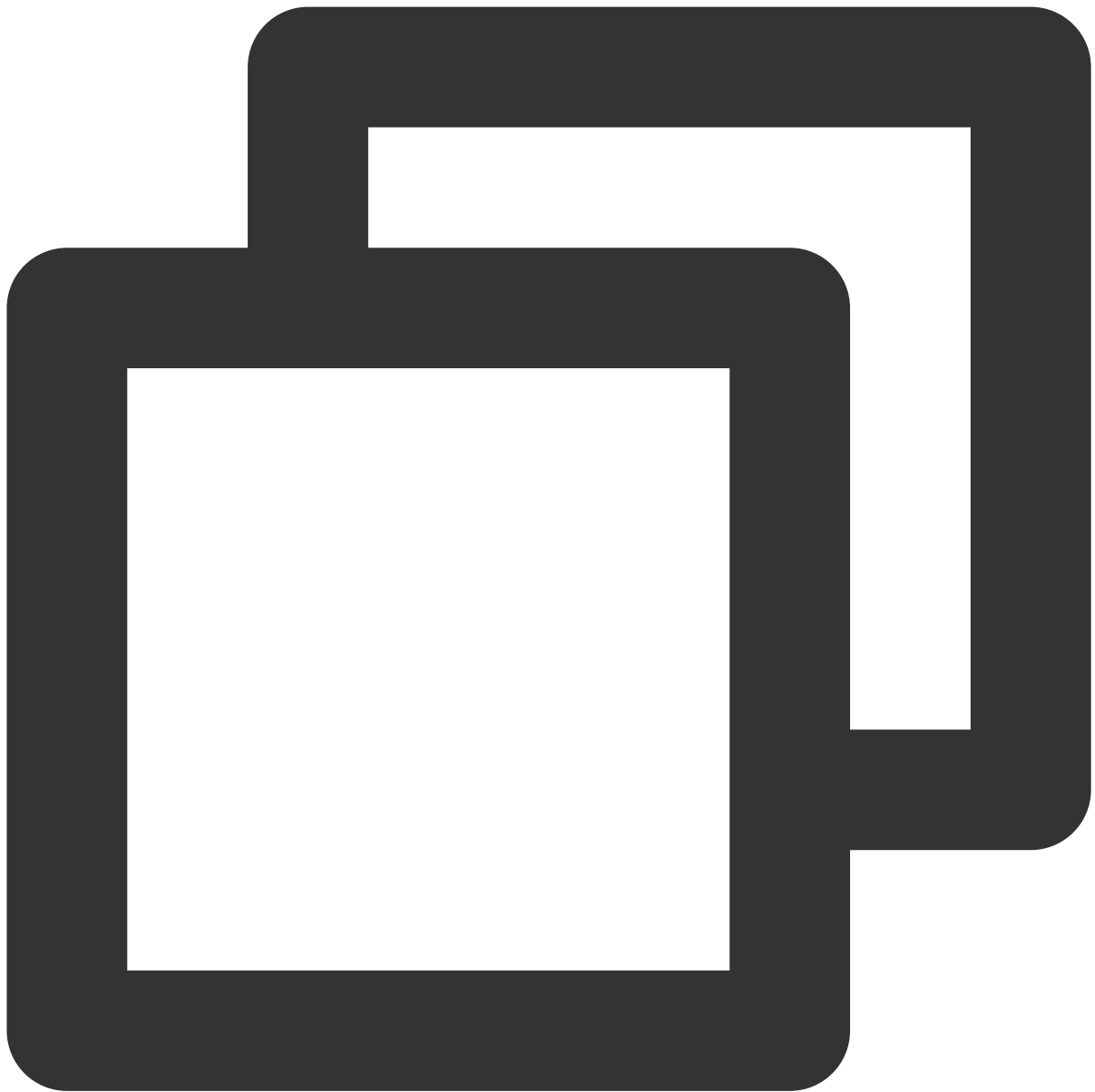
MiniStartOptions



```
class MiniStartOptions {  
    /// 小程序的进入路径
```

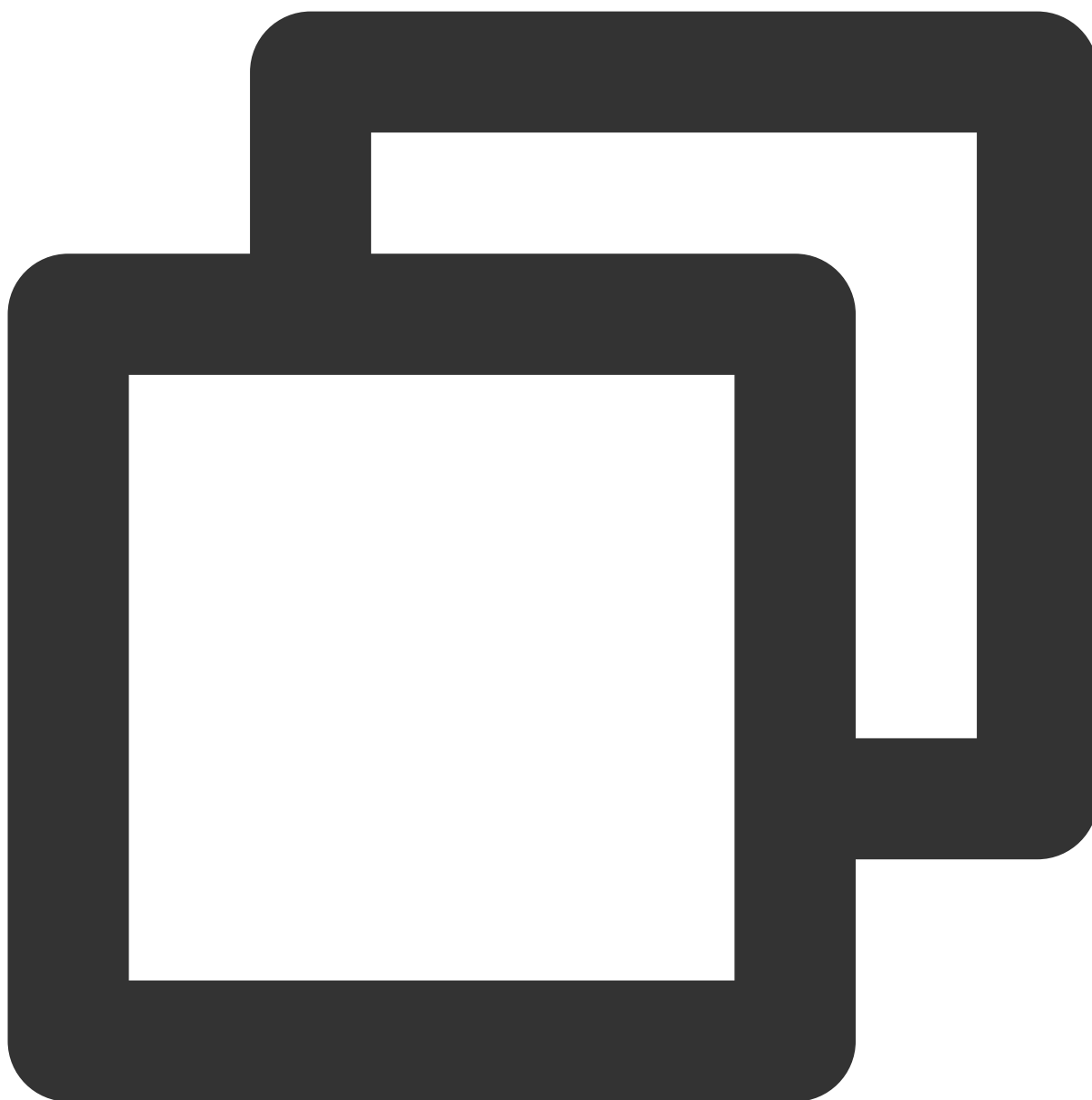
```
String? entryPath;  
  
/// 启动时始终更新小程序  
bool? isForceUpdate;  
  
/// 启动时传递给小程序的字符串参数  
String? params;  
}
```

ScanResult



```
class ScanResult {  
    /// qrcode 或条形码的结果字符串包含  
    String? result;  
  
    /// 代码类型  
    String? scanType;  
  
    /// 结果字符串的字符编码  
    String? charset;  
}
```

AppInfo



```
class AppInfo {  
    /// 小程序id  
    String appId;  
  
    /// 小程序包类型（发行版、开发版等）。请参阅 [AppVerType]  
    int appVerType;  
  
    /// 小程序版本
```

```
String version;

/// 小程序名称
String? name;

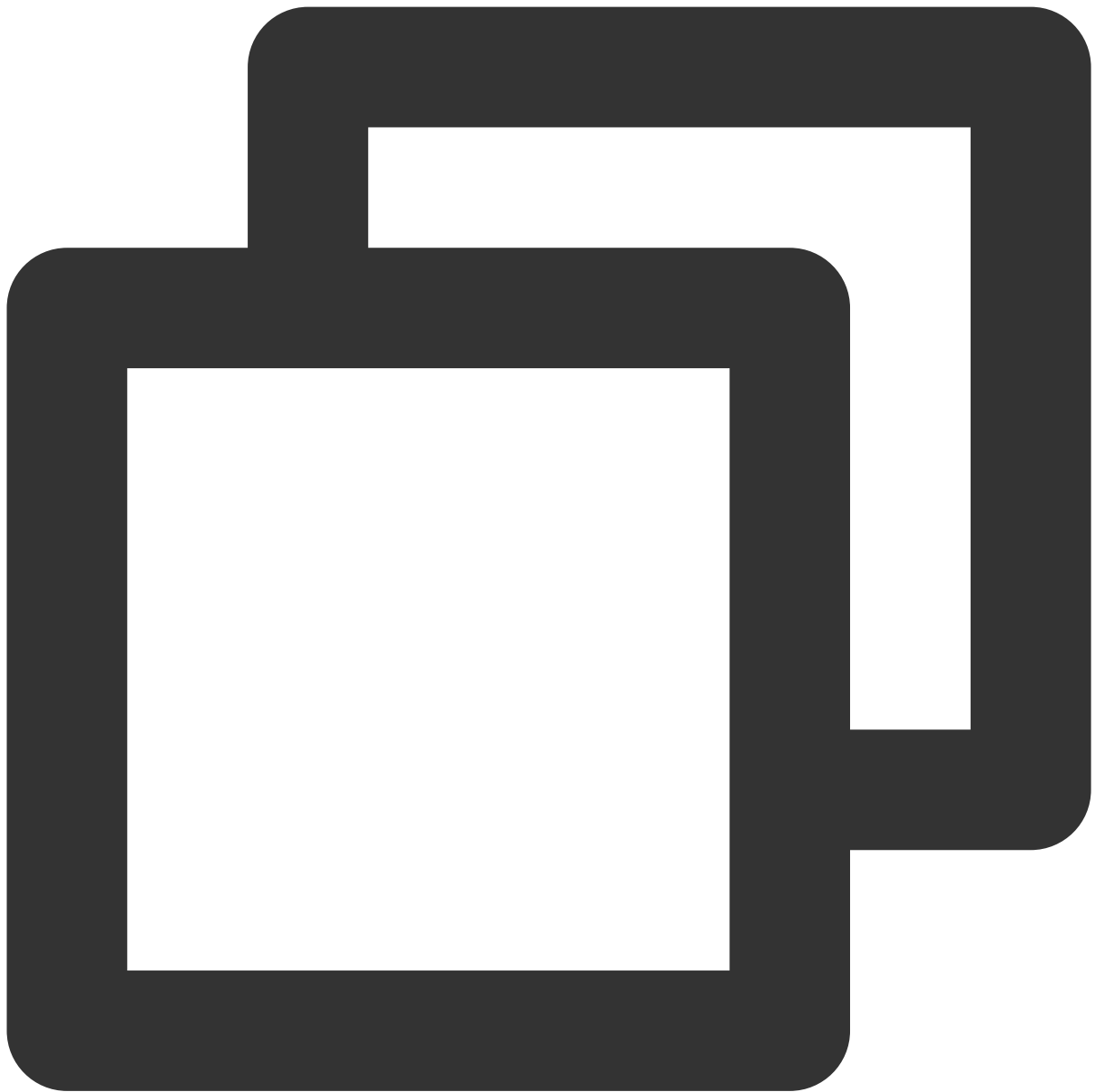
/// 小程序 icon 链接
String? iconUrl;

/// 小程序说明
String? appIntro;

/// 小程序开发员
String? appDeveloper;

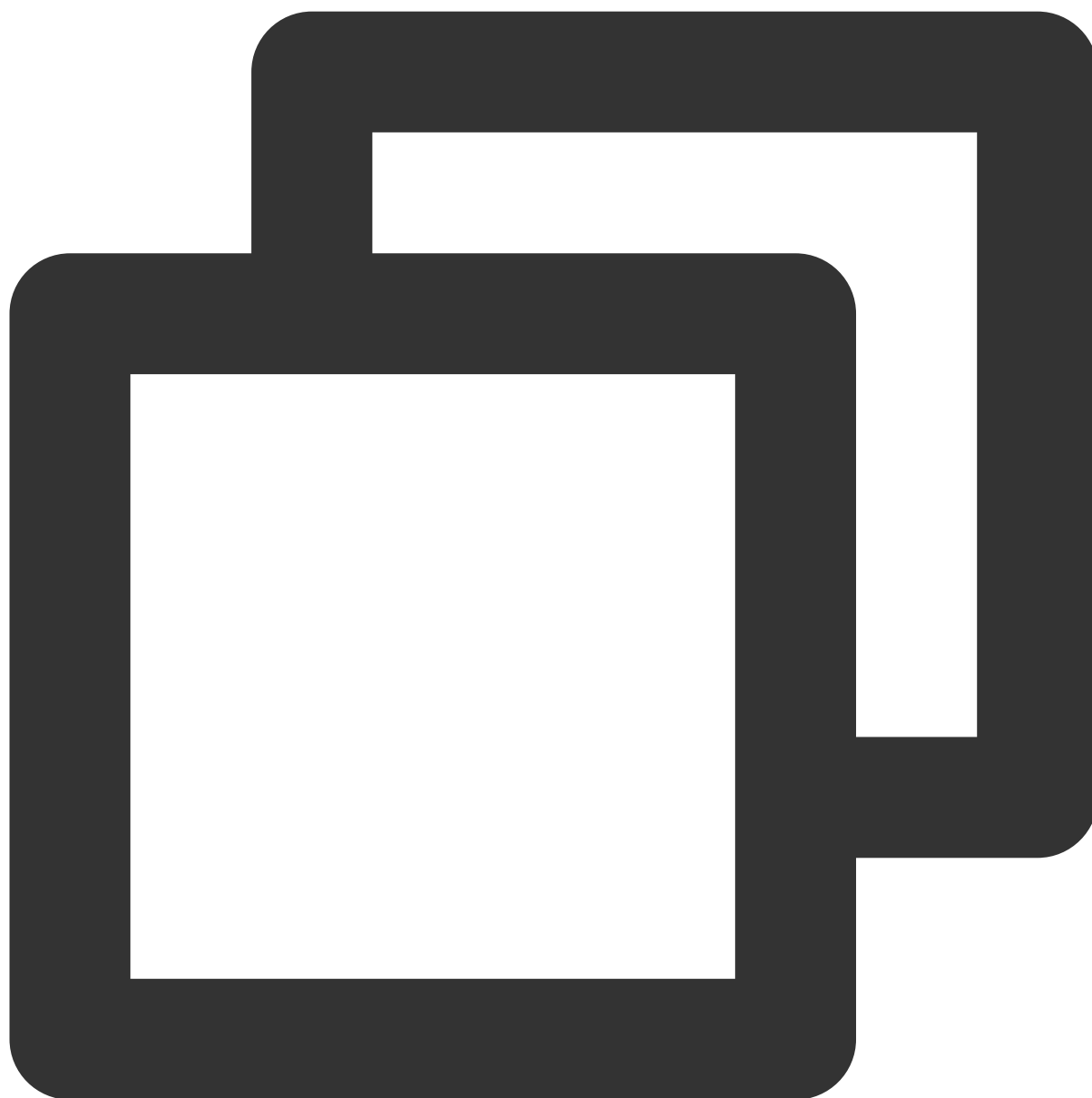
/// 小程序发布时间
int time;
}
```

AppVerType



```
/// 小程序软件包类型的常数，查看 [AppInfo]
class AppVerType {
    static const int online = 0;
    static const int develop = 1;
    static const int preview = 2;
    static const int experience = 3;
}
```

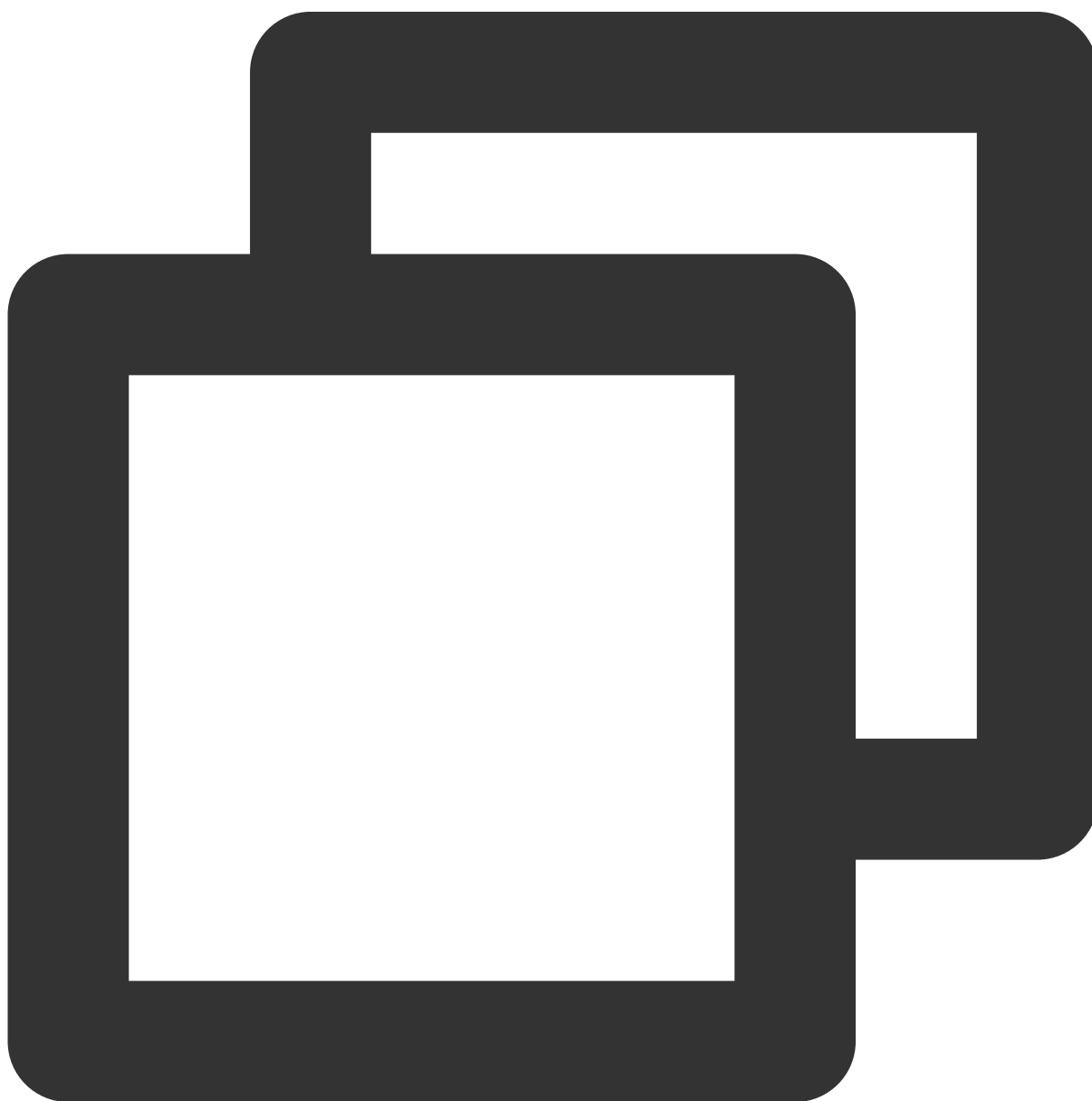
AccountInfo



```
class AccountInfo {  
    /// 活期账户的唯一 ID  
    String? uid;  
  
    /// 头像的链接  
    String? avatarUrl;
```

```
/// 往来帐户名称  
String? accountName;  
}
```

OpenApiHandler



```
abstract class OpenApiHandler {
```

```
//当小程序调用 wx.requestPayment, 请求第三方支付操作时调用

Future<Map<String, dynamic>> requestPayment(
    AppInfo appInfo, Map<Object?, Object?> params);

/// 当小程序调用 wx.getUserProfile 请求主机应用程序的用户信息时调用
///
Future<Map<String, dynamic>> getUserProfile(
    AppInfo appInfo, Map<Object?, Object?> params);

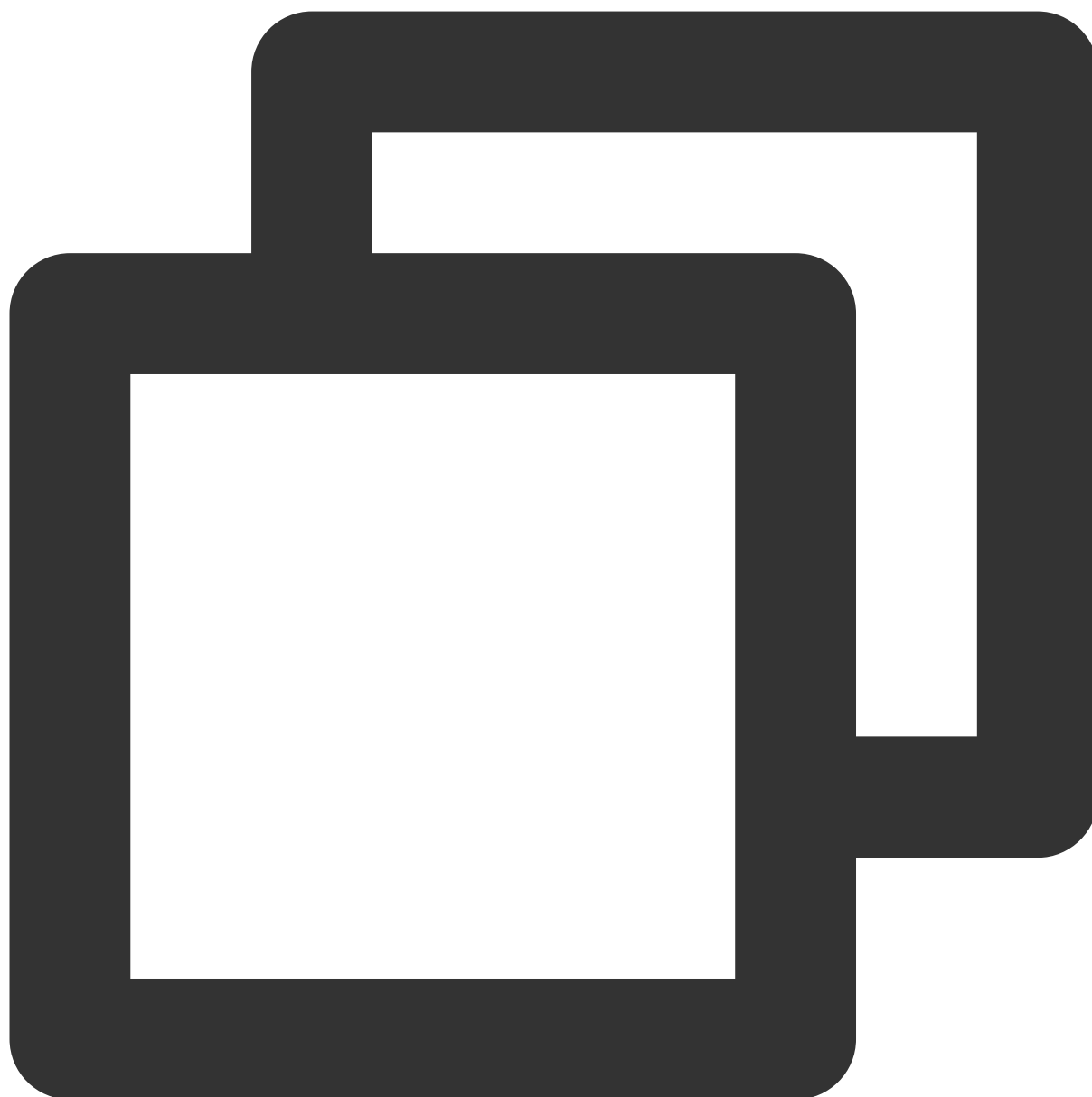
/// 当小程序调用 wx.login 时调用, 请求主机应用程序的登录证书
///
Future<Map<String, dynamic>> login(
    AppInfo appInfo, Map<Object?, Object?> params);

/// 当小程序调用 wx.checkSession 时调用, 请求主机应用程序的登录状态
/// 检查登录是否已过期
///
Future<Map<String, dynamic>> checkSession(
    AppInfo appInfo, Map<Object?, Object?> params);

/// 当小程序调用 wx.getUserInfo 时调用, 已被 wx.getUserProfile 过时
/// 兼容早期的小程序 api
///
Future<Map<String, dynamic>> getUserInfo(
    AppInfo appInfo, Map<Object?, Object?> params);

/// 当小程序调用 wx.getPhoneNumber 时调用, 获取当前用户的电话号码
///
Future<Map<String, dynamic>> getPhoneNumber(
    AppInfo appInfo, Map<Object?, Object?> params);
}
```

TcmppPlatformEventHandler



```
abstract class TcmppPlatformEventHandler {  
  
    Future<String> getAppName() async {  
        return "";  
    }  
  
    Future<String> getAppVersion() async {  
        return "";  
    }  
}
```

```
Future<List<CustomMenu>> getCustomMenus() async {  
    return [];  
}  
  
Future<void> customMenuClick(String menuId, ShareData? shareMenu) async {}  
  
Future<void> onMiniProgramStateChange(  
    String appId, MiniProgramState state) async {}  
  
Future<bool> reportEvent(int eventId, String eventName, AppInfo appInfo,  
    Map<Object?, Object?> params) async {  
    return false;  
}  
}
```

扩展插件

最近更新时间：2024-08-06 15:20:02

在开发、使用中，很多需要系统授权的API需要在iOS项目的info.plist文件和Android的AndroidManifest.xml文件中进行预置授权，但您的App可能并不需要此功能，所以tcmpp_flutter拆分除了扩展SDK，这样免去了不必要的授权，也缩小了核心模块的体积。

tcmpp_flutter 提供核心模块及扩展模块，方便使用者根据自己的情况进行接入。

某些 TCMPP 小程序 API 可能需要额外的隐私和权限。要使用这些 API，需要额外的 Flutter 插件依赖。

APIs	插件名	所需权限
LBS 相关 API（位置和 POI 搜索）	tcmpp_flutter_lbs	访问位置
MDNS APIs	tcmpp_flutter_mdns	访问本地网络
TCP/UDP APIs	tcmpp_flutter_network	访问网络
媒体应用程序接口（照片和视频）	tcmpp_flutter_media	访问图片库
无线应用程序接口、蓝牙应用程序接口、日历应用程序接口、联系人应用程序接口、剪贴板应用程序接口、生物识别验证应用程序接口	tcmpp_flutter_device	无线应用程序接口、蓝牙应用程序接口、日历应用程序接口、联系人应用程序接口、剪贴板应用程序接口、生物识别验证应用程序接口
地图 APIs	tcmpp_flutter_googlemap (for general) tcmpp_flutter_petalmap (for Huawei device)	访问位置

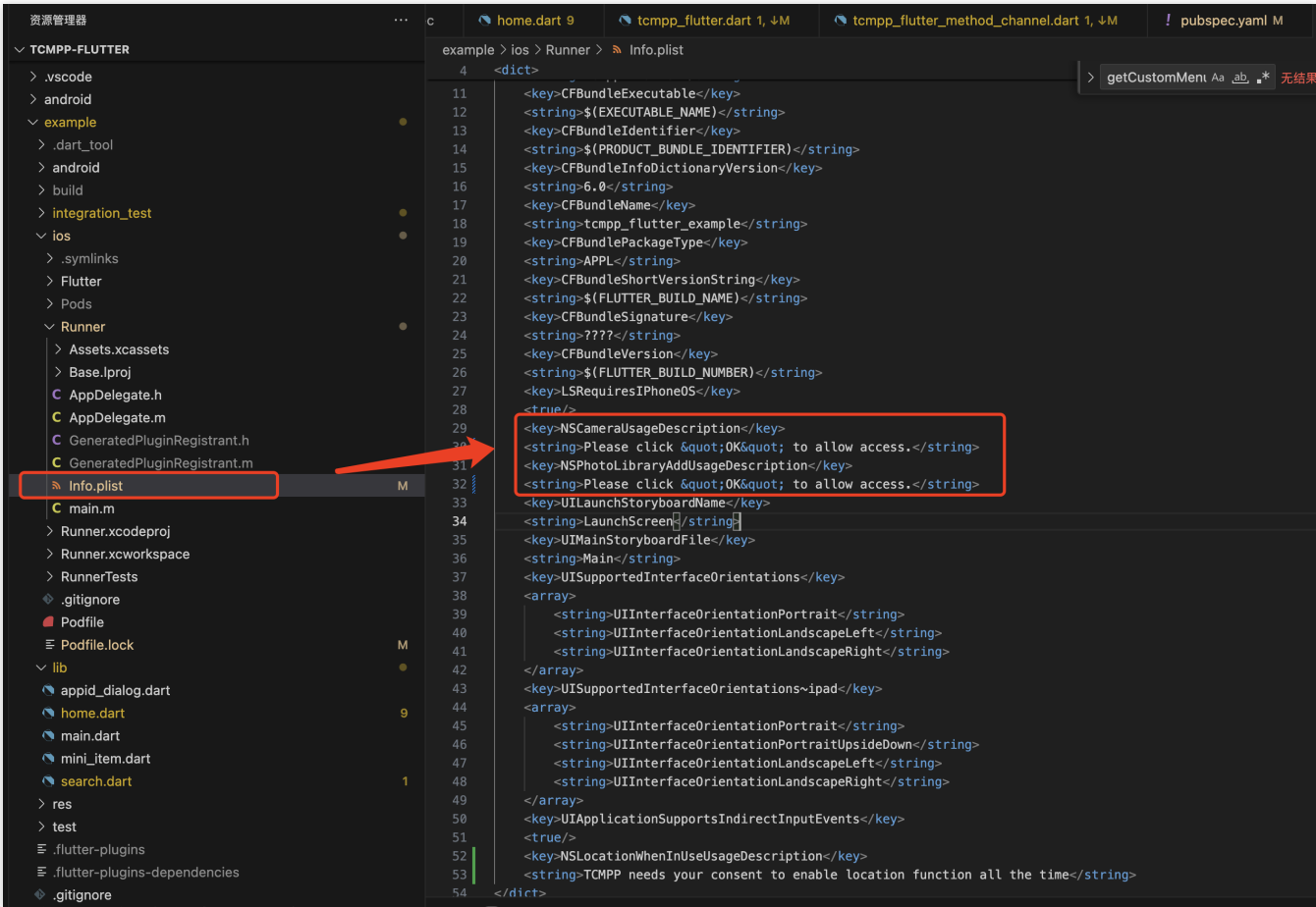
这些插件必须与 tcmpp_flutter 插件一起使用。只有 Android 平台需要地图 API 支持，iOS 默认包含这些 API。

Flutter 隐私合规

最近更新时间：2024-08-06 15:19:26

说明：

iOS 需要在 plist 文件中添加相应的权限说明。



iOS 插件名称和相应的权限信息：

插件名	涉及的系统权限
tcmpp_flutter_lbs	NSLocationAlwaysUsageDescription NSLocationWhenInUseUsageDescription NSLocationAlwaysAndWhenInUseUsageDescription NSLocationUsageDescription NSMotionUsageDescription
tcmpp_flutter_mdns	NSBonjourServices
tcmpp_flutter_network	NSAppTransportSecurity
tcmpp_flutter_media/tcmpp_flutter	NSPhotoLibraryUsageDescription NSCameraUsageDescription

tcmpp_flutter_device	NSCalendarsUsageDescription NSRemindersUsageDescription NSContactsUsageDescription NSFaceIDUsageDescription NSBluetoothAlwaysUsageDescription NSBluetoothPeripheralUsageDescription
----------------------	--

说明：

Android 需要在 android/app/src/main/AndroidManifest.xml 文件中添加相应的权限说明。

Android 插件名称和相应的权限信息：

插件名	涉及的系统权限
tcmpp_flutter	android.permission.CAMERA android.permission.WRITE_EXTERNAL_STORAGE android.permission.FLASHLIGHT
tcmpp_flutter_lbs	android.permission.ACCESS_FINE_LOCATION
tcmpp_flutter_network	android.permission.INTERNET android.permission.ACCESS_NETWORK_STATE
tcmpp_flutter_media	无
tcmpp_flutter_device	android.permission.BLUETOOTH_ADMIN android.permission.BLUETOOTH_SCAN android.permission.BLUETOOTH_ADVERTISE android.permission.BLUETOOTH_CONNECT android.permission.WRITE_CONTACTS android.permission.READ_CONTACTS android.permission.USE_FINGERPRINT android.permission.USE_BIOMETRIC android.permission.READ_CALENDAR android.permission.WRITE_CALENDAR android.permission.ACCESS_FINE_LOCATION android.permission.ACCESS_WIFI_STATE android.permission.CHANGE_WIFI_STATE android.permission.ACCESS_NETWORK_STATE android.permission.CHANGE_NETWORK_STATE
tcmpp_flutter_googlemap	android.permission.ACCESS_FINE_LOCATION
tcmpp_flutter_petalmap	android.permission.INTERNET android.permission.ACCESS_NETWORK_STATE android.permission.CHANGE_WIFI_STATE

android.permission.ACCESS_COARSE_LOCATION
android.permission.ACCESS_FINE_LOCATION